

# Algorithms Lab

Alberto Montes  
*malberto@student.ethz.ch*

January 22, 2017

# Contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Fundamentals</b>   | <b>3</b>  |
| 1.1      | Introduction . . . . .  | 3         |
| 1.2      | Binary Search, Sliding Window, Graph Traversals, Greedy . . . . . | 10        |
| 1.3      | CGAL . . . . .  | 16        |
| 1.4      | BGL . . . . .   | 24        |
| <b>2</b> | <b>Advanced Algorithms</b>  | <b>34</b> |
| 2.1      | Dynamic Programming, Brute Force, Split and List . . . . .        | 34        |
| 2.2      | BGL Flows . . . . .   | 45        |
| 2.3      | Linear/Quadratic Programing . . . . .                             | 53        |
| 2.4      | Proximity Structures in CGAL . . . . .                            | 61        |
| 2.5      | Advanced Flows . . . . .  | 69        |
| <b>3</b> | <b>Exam Preparation</b>   | <b>79</b> |

# Chapter 1

## Fundamentals

### 1.1 Introduction

#### Build the sum

*Keywords—*

```
1  #include <vector>
2  #include <iostream>
3
4  using namespace std;
5
6  void do_sum(){
7      int n; cin>> n;
8      float sum = 0.0;
9      float sumant;
10     for (int i =0; i < n; i++) {
11         cin >> sumant;
12         sum += sumant;
13     }
14
15     cout << sum << endl;
16 }
17
18 int main() {
19     int T; cin >> T;
20     for (int t=0; t < T; t++){
21         do_sum();
22     }
23 }
```

## Even Pairs

*Keywords*— Precompute

```
1  #include <vector>
2  #include <iostream>
3
4  using namespace std;
5
6  void do_even_count(){
7      int n; cin >> n;
8      vector<int> bits(n);
9      for (int i =0; i < n; i++)
10         cin >> bits [i];
11
12     // Precompute Si's
13     vector<int> partial_sum(n);
14     partial_sum[0] = bits[0];
15     for (int i = 1; i < n; i++) {
16         partial_sum[i] = partial_sum[i-1] + bits[i];
17     }
18
19     vector<int> evens(n), odds(n);
20     evens[0] = 0; odds[0] = 0;
21     if (bits[0] == 0)
22         evens[0] = 1;
23     else
24         odds[0] = 1;
25     for (int i = 1; i < n; i++) {
26         evens[i] = evens[i-1];
27         odds[i] = odds[i-1];
28         if (partial_sum[i] % 2 == 0)
29             evens[i]++;
30         else
31             odds[i]++;
32     }
33
34     // Calculate the result
35     int counter = 0;
36     if (bits[0] == 0)
37         counter++;
38     for (int i =1; i < n; i++){
39         if (partial_sum[i] % 2 == 0)
40             counter += evens[i-1] + 1;
41         else
42             counter += odds[i-1];
43     }
44     cout << counter << endl;
45 }
46
47
48 int main() {
49     int T; cin >> T;
50     for (int t=0; t < T; t++){
51         do_even_count();
52     }
53 }
```

# Dominoes

## *Keywords*—

```
1  #include <vector>
2  #include <iostream>
3
4  using namespace std;
5
6  void compute_dominoes_falling(){
7      int n; cin >> n;
8      vector<unsigned long int> dominoes_height(n);
9      for (int i =0; i < n; i++)
10         cin >> dominoes_height[i];
11
12     int max = 1;
13
14     for (int i = 0; i < n; i++) {
15         if (i + dominoes_height[i] > max)
16             max = i + dominoes_height[i];
17         else if (max == i + 1)
18             break;
19     }
20     if (max > n)
21         max = n;
22
23     cout << max << endl;
24 }
25
26 int main() {
27     ios_base::sync_with_stdio(false);
28     int T; cin >> T;
29     for (int t=0; t < T; t++){
30         compute_dominoes_falling();
31     }
32 }
```

## Even Matrices

*Keywords*— Precompute

```
1 #include <vector>
2 #include <iostream>
3
4 using namespace std;
5
6
7 // Implementation in  $O(n^3)$  time
8 void do_even_count_matrices(){
9     int n; cin >> n;
10    vector<vector<int>> > M(n, vector<int>(n));
11    for (int i = 0; i < n; i++)
12        for (int j = 0; j < n; j++)
13            cin >> M[i][j];
14
15    // Precompute pMi's
16    vector<vector<int>> > pM(n+1, vector<int>(n+1));
17    for (int i = 0; i <= n; i++) {
18        pM[i][0] = 0;
19        pM[0][i] = 0;
20    }
21
22    for (int i = 1; i <= n; i++) {
23        for (int j = 1; j <= n; j++) {
24            pM[i][j] = pM[i-1][j] + pM[i][j-1] - pM[i-1][j-1] + M[i-1][j-1];
25        }
26    }
27
28    int counter = 0;
29    for (int i1 = 1; i1 <= n; i1++) {
30        for (int i2 = i1; i2 <= n; i2++) {
31            // We reduce the problem to one dimension
32            vector<int> S(n+1); // Do even pairs on array S
33            vector<int> pS(n+1); // pS contains partial sums of S
34            pS[0] = 0;
35            for (int k = 1; k <= n; k++) {
36                S[k] = pM[i2][k] - pM[i2][k-1] - pM[i1-1][k] + pM[i1-1][k-1];
37                pS[k] = pS[k-1] + S[k];
38            }
39
40            // Do even pairs  $O(n)$  algorithm on array S
41            int onedim_sol = 0;
42            int even = 0, odd = 0;
43            for (int j = 1; j <= n; j++) {
44                // even = # of partial sums of array (S[1], ..., S[j-1]) that are even.
45                // odd = # of partial sums of array (S[1], ..., S[j-1]) that are odd.
46                if (pS[j] % 2 == 0){
47                    onedim_sol += even + 1;
48                    even++;
49                }
50                else {
51                    onedim_sol += odd;
52                    odd++;
53                }
54            }
55            counter += onedim_sol;
56        }
57    }
58
59    cout << counter << endl;
60 }
61
62 int main() {
63     ios_base::sync_with_stdio(false);
64     int T; cin >> T;
65     for (int t=0; t < T; t++){
66         do_even_count_matrices();
67     }
68 }
```

## False Coin

*Keywords*— Tree Search

```
1  #include <vector>
2  #include <iostream>
3
4  using namespace std;
5
6  void test_false_coin() {
7      int n; cin >> n;
8
9      vector<int> candidates(n, 0);
10     vector<int> lw_coins(n, 0), hw_coins(n, 0);
11
12     int num_trials; cin >> num_trials;
13
14     for (int i = 0; i < num_trials; i++) {
15         int num_per_side; cin >> num_per_side;
16
17         vector<int> left_coins(n, false), right_coins(n, false);
18         int left_pos, right_pos;
19         for (int j = 0; j < num_per_side; j++) {
20             cin >> left_pos;
21             left_coins[left_pos-1] = true;
22         }
23         for (int j = 0; j < num_per_side; j++) {
24             cin >> right_pos;
25             right_coins[right_pos-1] = true;
26         }
27
28         char comparison; cin >> comparison;
29
30         // For each trial eliminate all the possible candidates depending where on the balance
31         // appear
32         for (int j = 0; j < n; j++) {
33             switch (comparison) {
34                 case '=': {
35                     if (left_coins[j] || right_coins[j])
36                         candidates[j] = -1;
37                     else if (candidates[j] == 0)
38                         candidates[j] = 1;
39                     break;
40                 }
41                 case '<': {
42                     if (left_coins[j]) {
43                         lw_coins[j]++;
44                         if (hw_coins[j] > 0) {
45                             candidates[j] = -1;
46                         } else if (candidates[j] == 0) {
47                             candidates[j] = 1;
48                         }
49                     } else if (right_coins[j]) {
50                         hw_coins[j]++;
51                         if (lw_coins[j] > 0) {
52                             candidates[j] = -1;
53                         } else if (candidates[j] == 0) {
54                             candidates[j] = 1;
55                         }
56                     } else {
57                         candidates[j] = -1;
58                     }
59                     break;
60                 }
61                 case '>': {
62                     if (right_coins[j]) {
63                         lw_coins[j]++;
64                         if (hw_coins[j] > 0) {
65                             candidates[j] = -1;
66                         } else if (candidates[j] == 0) {
67                             candidates[j] = 1;
68                         }
69                     } else if (left_coins[j]) {
70                         hw_coins[j]++;
71                         if (lw_coins[j] > 0) {
72                             candidates[j] = -1;
73                         } else if (candidates[j] == 0) {
74                             candidates[j] = 1;
75                         }
76                     } else {
77                         candidates[j] = -1;
```

```

78         }
79         break;
80     }
81     default: break;
82 }
83 }
84 }
85
86 int count_candidates = 0;
87 int coin_id = 0;
88 for (int i = 0; i < n; i++) {
89     if (candidates[i] == 1) {
90         count_candidates++;
91         coin_id = i + 1;
92     }
93 }
94 // If there is more than one candidate there is not an unique solution
95 if (count_candidates > 1) {
96     coin_id = 0;
97 }
98 cout << coin_id << endl;
99
100 }
101
102 int main() {
103     ios_base::sync_with_stdio(false);
104     int T; cin >> T;
105     for (int t=0; t < T; t++){
106         test_false_coin();
107     }
108 }

```



## Deck of Cards

*Keywords*— Sliding Window

```
1
2 #include <vector>
3 #include <iostream>
4 #include <limits>
5 #include <stdlib.h>
6
7 using namespace std;
8
9 // Implementation in O(n)
10 void deck_of_cards(){
11     long n; cin >> n;
12     long k; cin >> k;
13
14     vector<long> values(n);
15     for (int i = 0; i < n; i++) {
16         cin >> values[i];
17     }
18
19     long i_r = 0, j_r = 0;
20     long start = 0, end = 0;
21     long sum = values[0];
22     long min_value = abs(k-sum);
23
24     // Slide a window and keep track of the minimum difference between k and the sum of values
25     while (start < n) {
26         if (sum <= k) {
27             if (end < n-1) {
28                 sum += values[++end];
29             } else { break; }
30         } else {
31             sum -= values[start++];
32         }
33         if (abs(k-sum) < min_value) {
34             min_value = abs(k-sum);
35             i_r = start;
36             j_r = end;
37         }
38     }
39
40     cout << i_r << ' ' << j_r << endl;
41 }
42
43 int main() {
44     ios_base::sync_with_stdio(false);
45     int T; cin >> T;
46     for (int t=0; t < T; t++){
47         deck_of_cards();
48     }
49 }
```

## 1.2 Binary Search, Sliding Window, Graph Traversals, Greedy

### Search Snippets

**Keywords**— Sliding Window

```
1  /*
2  Implementation sliding window.
3  Hint: to improve the efficiency, instead of loop along all the appearance vector to see if all the
4  words appear, keep up a counter and changing it with conditions. It is much more efficient and
      clean
5  */
6  #include <vector>
7  #include <iostream>
8  #include <algorithm>
9  #include <math.h>
10 #include <limits.h>
11
12 using namespace std;
13
14 struct word {
15     int id;
16     int position;
17 };
18
19 void search_snippets() {
20     int n; cin >> n;
21
22     vector<int> m(n);
23     int t = 0; // total word appearance
24     for (int i = 0; i < n; i++) {
25         cin >> m[i];
26         t += m[i];
27     }
28
29     vector<word> w; w.reserve(t);
30     for (int i = 0; i < n; i++) {
31         for (int j = 0; j < m[i]; j++) {
32             word wo; wo.id = i;
33             cin >> wo.position;
34             w.push_back(wo);
35         }
36     }
37     sort(w.begin(), w.end(), [](word a, word b) {
38         return a.position < b.position;
39     });
40
41     vector<int> a(n, 0);
42     int min_d = w[t-1].position - w[0].position + 1;
43     int s = 0, e = 0, u = n; // start, end, and uncovered counter
44     a[w[s].id]++; u--;
45     // Sliding Window
46     while (e != t) {
47         while (e < t - 1 && u > 0) {
48             e++;
49             if (a[w[e].id] == 0) u--;
50             a[w[e].id]++;
51         }
52         if (u != 0) break;
53         do {
54             a[w[s].id]--;
55             if (a[w[s].id] == 0) ++u;
56         } while (++s != e && u == 0); // Even you go out of the while loop, s will have been
57                                     // increased undiredbly
58         min_d = min(min_d, w[e].position - w[s-1].position + 1);
59     }
60
61     cout << min_d << endl;
62 }
63
64 int main() {
65     ios_base::sync_with_stdio(false);
66     int T; cin >> T;
67     for (int t=0; t < T; t++){
68         search_snippets();
69     }
70 }
```

# Boats

**Keywords**— Interval Scheduling

```
1  /*
2  Implementation using two fors to iterate
3
4  Correct
5  */
6  #include <vector>
7  #include <iostream>
8  #include <algorithm>
9
10 using namespace std;
11
12 void boats() {
13     long n; cin >> n;
14
15     vector<pair<long, long> > boats(n);
16     for (int i = 0; i < n; i++) {
17         cin >> boats[i].second;
18         cin >> boats[i].first;
19     }
20
21     sort(boats.begin(), boats.end(), [](pair<long, long> a, pair<long, long> b) {
22         return a.first < b.first;
23     });
24
25     long current_last_position = boats[0].first;
26     long number_boats = 1;
27     long i = 1;
28     while (i < n) {
29         if (current_last_position > boats[i].first) {
30             i++;
31         } else if (boats[i].first - current_last_position > boats[i].second) {
32             number_boats++;
33             current_last_position = boats[i].first;
34             i++;
35         } else {
36             long first_end = boats[i].second + current_last_position;
37             long min_position = i;
38             for (long j = i+1; j < n; j++) {
39                 long end_position;
40                 if (boats[j].first - current_last_position > boats[j].second) {
41                     end_position = boats[j].first;
42                 } else {
43                     end_position = current_last_position + boats[j].second;
44                 }
45                 if (end_position < first_end) {
46                     first_end = end_position;
47                     min_position = j;
48                 }
49             }
50             number_boats++;
51             current_last_position = first_end;
52             i = min_position + 1;
53         }
54     }
55
56     cout << number_boats << endl;
57
58 }
59
60
61 int main() {
62     ios_base::sync_with_stdio(false);
63     int T; cin >> T;
64     for (int t=0; t < T; t++){
65         boats();
66     }
67 }
```

# Moving Books

*Keywords*— Greedy

```
1 #include <vector>
2 #include <set>
3 #include <iostream>
4 #include <algorithm>
5
6 using namespace std;
7
8
9 void moving_books() {
10     long n; cin >> n;
11     long m; cin >> m;
12
13     vector<long> s(n), w(m);
14     for (int i = 0; i < n; i++) {
15         cin >> s[i];
16     }
17     for (int i = 0; i < m; i++) {
18         cin >> w[i];
19     }
20
21     // Check if the heaviest box can not be brought by nobody
22     int max_s = *max_element(s.begin(), s.end());
23     int max_w = *max_element(w.begin(), w.end());
24     if (max_w > max_s) {
25         cout << "impossible\n";
26         return;
27     }
28
29     multiset<int, greater<int> > ws;
30     sort(s.begin(), s.end(), greater<int>());
31
32     for (int i = 0; i < m; i++) ws.insert(w[i]);
33     int r = 0;
34     while (!ws.empty()) {
35         r++;
36         for (int i = 0; i < n; i++) {
37             auto j = ws.lower_bound(s[i]);
38             if (j != ws.end()) {
39                 ws.erase(j);
40             } else {
41                 break;
42             }
43         }
44     }
45
46     cout << 3*r-1 << endl;
47 }
48
49 int main() {
50     ios_base::sync_with_stdio(false);
51     int T; cin >> T;
52     for (int t=0; t < T; t++){
53         moving_books();
54     }
55 }
```

## Evolution

*Keywords*— DFS, Binary Search

```
1  #include <vector>
2  #include <map>
3  #include <iostream>
4  #include <string>
5  #include <algorithm>
6  #include <unordered_map>
7
8  using namespace std;
9
10
11 // Binary Search
12 int binary(int b, vector<int>& path, vector<int>& age) {
13     int l = 0; int r = path.size()-1;
14     while(l != r){
15         int m = (l+r)/2;
16         if (age[path[m]] > b) l = m+1; else r = m;
17     }
18     return path[l];
19 }
20
21 // DFS
22 void dfs(int u, vector<vector<int> >& tree, vector<int>& path, vector<vector<pair<int,int> > >&
    query, vector<int>& result, vector<int>& age) {
23     // process queries
24     for (int i = 0; i < query[u].size(); i++) {
25         result[query[u][i].second] = binary(query[u][i].first, path, age);
26     }
27
28     // continue search
29     for (int i = 0; i < tree[u].size(); ++i){
30         int v = tree[u][i];
31         path.push_back(v);
32         dfs(v, tree, path, query, result, age);
33     }
34     path.pop_back();
35 }
36
37 void evolution() {
38     int n, q; cin >> n >> q;
39
40     // Store the names and respective ages
41     unordered_map<string, int> species_to_index;
42     vector<string> species(n);
43     vector<int> age(n);
44     string name;
45     for (int i = 0; i < n; i++) {
46         cin >> name;
47         species_to_index[name] = i;
48         species[i] = name;
49         cin >> age[i];
50     }
51
52     // Find root
53     int root = max_element(age.begin(), age.end()) - age.begin();
54
55     // Read tree
56     vector<vector<int> > tree(n);
57     string child, parent;
58     for (int i = 0; i < n - 1; i++) {
59         cin >> child >> parent;
60         tree[species_to_index[parent]].push_back(species_to_index[child]);
61     }
62
63     // Read queries: for each species store a vector of queries consisting of the age b and the
64     // index of the query i
65     vector<vector<pair<int, int> > > query(n);
66     for (int i = 0; i < q; i++) {
67         cin >> name;
68         int b; cin >> b;
69         query[species_to_index[name]].push_back(make_pair(b, i));
70     }
71
72     // Process queries in one tree transversal
73     vector<int> path; path.push_back(root);
74     vector<int> result(q);
75     dfs(root, tree, path, query, result, age);
76 }
```

```
77 // Output result
78 for (int i = 0; i < q; i++) {
79     cout << species[result[i]];
80     if ( i < q - 1 ) cout << " ";
81 }
82 cout << endl;
83 }
84
85 int main() {
86     ios_base::sync_with_stdio(false);
87     int T; cin >> T;
88     for (int t=0; t < T; t++){
89         evolution();
90     }
91 }
```

# Octopussy

## Keywords— Greedy

```
1
2 #include <iostream>
3 #include <vector>
4 #include <queue>
5 #include <algorithm>
6
7 using namespace std;
8
9 long find_minimum_time(vector<long> &t, long pos) {
10     if (pos == 0) {
11         return t[pos];
12     }
13     long upper_pos = (pos % 2 == 1) ? (pos - 1) / 2 : (pos - 2) / 2;
14     long next_pos = (pos % 2 == 1) ? pos + 1 : pos - 1;
15     long min_upper_pos = find_minimum_time(t, upper_pos);
16     // Check if the time limit of the pair bomb is more limiting or not.
17     if (t[next_pos] <= min_upper_pos - 2) {
18         return min(t[pos], min_upper_pos - 1);
19     } else {
20         return min(t[pos], min_upper_pos - 2);
21     }
22 }
23
24 void octopussy() {
25     int n; cin >> n;
26
27     vector<long> t(n);
28     for (int i = 0; i < n; i++) {
29         cin >> t[i];
30     }
31
32     vector<bool> disarmed(n, false);
33
34     auto cmp = [&t](pair<long, long> left, pair<long, long> right) {
35         if ( left.second > right.second ) { return true; }
36         else if ( left.second == right.second ) { return t[left.first] > t[right.first]; }
37         else { return false; }
38     };
39     priority_queue<pair<long, long>, vector<pair<long, long> >, decltype(cmp)> bombs_queue(cmp);
40
41     for (long j = (n-3)/2+1; j < n; j++) {
42         long min_time = find_minimum_time(t, j);
43         bombs_queue.push(pair<long, long>(j, min_time));
44     }
45
46     long t_time = 0;
47     while( !bombs_queue.empty() ) {
48         auto p_bomb = bombs_queue.top();
49         bombs_queue.pop();
50         // The most priority bomb has expired its time...BOOOM!
51         long b_index = p_bomb.first;
52         if (t[b_index] <= t_time) { break; }
53         // Marked the bomb as disarmed
54         disarmed[b_index] = true;
55         t_time++;
56         /* Now see if the bomb next to it has been disarmed to let the bomb standing over them to
57         enter the priority queue. If not...too late...BOOOOOM!! */
58         if (b_index == 0) { continue; } // The bomb at index 0 has no bomb standing over it.
59         long next_b_index = (b_index % 2 == 1) ? b_index + 1 : b_index - 1;
60         if ( disarmed[next_b_index] ) {
61             long std_b_index = (b_index % 2 == 1) ? (b_index - 1) / 2 : (b_index - 2) / 2;
62             bombs_queue.push(pair<long, long>(std_b_index, find_minimum_time(t, std_b_index)));
63         }
64     }
65
66     bool all_disarmed = all_of(disarmed.begin(), disarmed.end(), [](bool v){return v;});
67     string response = (all_disarmed) ? "yes" : "no";
68     cout << response << endl;
69 }
70
71
72 int main() {
73     int t; cin >> t;
74     for (int i = 0; i < t; i++) {
75         octopussy();
76     }
77 }
```

## 1.3 CGAL

### Hit

*Keywords*— Intersection

```
1
2 #include <CGAL/Exact_predicates_exact_constructions_kernel_with_sqrt.h>
3 #include <iostream>
4
5 typedef CGAL::Exact_predicates_exact_constructions_kernel_with_sqrt K;
6
7 using namespace std;
8
9
10 void compute_hit(int n) {
11     long x, y, a, b;
12     cin >> x >> y >> a >> b;
13     K::Point_2 xy(x, y), ab(a, b);
14     K::Ray_2 ray(xy, ab);
15
16     bool hit = false;
17     for (int i = 0; i < n; i++) {
18         long r, s, t, u;
19         cin >> r >> s >> t >> u;
20         if (hit) {
21             continue;
22         }
23         K::Point_2 rs(r, s), tu(t, u);
24         K::Segment_2 seg(rs, tu);
25         if (CGAL::do_intersect(ray, seg)) {
26             hit = true;
27         }
28     }
29
30     string result = (hit) ? "yes" : "no";
31     cout << result << endl;
32 }
33
34
35
36 int main() {
37     ios_base::sync_with_stdio(false);
38     int nb_obstacles; cin >> nb_obstacles;
39
40     while(nb_obstacles > 0) {
41         compute_hit(nb_obstacles);
42         cin >> nb_obstacles;
43     }
44     return 0;
45 }
```



## First Hit

*Keywords*— Intersection

```
1  #include <CGAL/Exact_predicates_exact_constructions_kernel.h>
2  #include <vector>
3  #include <algorithm>
4  #include <type_traits>
5  #include <stdexcept>
6  #include <limits>
7
8  using namespace std;
9
10 typedef CGAL::Exact_predicates_exact_constructions_kernel K;
11 typedef result_of<K::Intersect_2(K::Ray_2,K::Segment_2)>::type IT;
12
13 // round down to next double
14 double floor_to_double(const K::FT& x) {
15     double a = floor(CGAL::to_double(x));
16     while (a > x) a -= 1;
17     while (a+1 <= x) a += 1;
18     return a;
19 }
20
21 // read segment cin; as each coordinate can be represented as a
22 // long, this is significantly faster than K::Segment_2 s; cin >> s;
23 inline K::Segment_2 read_segment() {
24     long x1, y1, x2, y2;
25     cin >> x1 >> y1 >> x2 >> y2;
26     return K::Segment_2(K::Point_2(x1,y1), K::Point_2(x2,y2));
27 }
28
29 // clip/set target of s to o
30 void shorten_segment(K::Segment_2& s, const IT& o) {
31     if (const K::Point_2* p = boost::get<K::Point_2>(&*o))
32         s = K::Segment_2(s.source(), *p);
33     else if (const K::Segment_2* t = boost::get<K::Segment_2>(&*o))
34         // select endpoint of *t closer to s.source()
35         if (CGAL::collinear_are_ordered_along_line (s.source(), t->source(), t->target()))
36             s = K::Segment_2(s.source(), t->source());
37         else
38             s = K::Segment_2(s.source(), t->target());
39     else
40         throw runtime_error("Strange segment intersection.");
41 }
42
43 void find_hit(size_t n) {
44     // read input
45     K::Ray_2 r;
46     cin >> r;
47     vector<K::Segment_2> segs;
48     segs.reserve(n);
49     for (size_t i = 0; i < n; ++i) segs.push_back(read_segment());
50     random_shuffle(segs.begin(), segs.end());
51
52     // clip the ray at each segment hit (cuts down on the number of
53     // intersection points to be constructed: for a uniformly random
54     // order of segments, the expected number of constructions is
55     // logarithmic in the number of segments that intersect the initial // ray.)
56     K::Segment_2 rc(r.source(), r.source());
57
58     // find some segment hit by r
59     for (size_t i; i < n; ++i)
60         if (CGAL::do_intersect(segs[i], r)) {
61             shorten_segment(rc, CGAL::intersection(segs[i], r));
62             break;
63         }
64
65     if (i == n) { cout << "no\n"; return; }
66     // check remaining segments against rc
67     while (++i < n)
68         if (CGAL::do_intersect(segs[i], rc))
69             shorten_segment(rc, CGAL::intersection(segs[i], r)); // not rc!
70
71     cout << floor_to_double(rc.target().x()) << " " << floor_to_double(rc.target().y()) << "\n";
72 }
73
74 int main() {
75     // sanity check
76     if (numeric_limits<long>::digits < 51)
77
```

```
78         throw runtime_error("long has <51 bits mantissa");
79     ios_base::sync_with_stdio(false);
80     cout << setiosflags(ios::fixed) << setprecision(0);
81     for (size_t n; cin >> n && n > 0;)
82         find_hit(n);
83     return 0;
84 }
```

# Hiking Maps

**Keywords**— Triangle Contains, Sliding Window

```
1 #include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
2 #include <iostream>
3 #include <vector>
4 #include <algorithm>
5
6 typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
7 typedef K::Point_2 P;
8 typedef std::vector<P> Pts;
9 typedef std::vector<int> Covered;
10 typedef std::vector<Covered> Trs;
11 typedef Covered::const_iterator Iterator;
12
13 using namespace std;
14
15 // Function to check whether a point is inside a triangle
16 inline bool contains(const Pts& t, const P& p) {
17     return !CGAL::right_turn(t[0],t[1],p) && !CGAL::right_turn(t[2],t[3],p) && !CGAL::right_turn(t
18         [4],t[5],p);
19 }
20
21 void hiking_maps() {
22     int m, n; cin >> m >> n;
23
24     // Read the path
25     Pts path(m);
26     long x, y;
27     for (int i = 0; i < m; i++) {
28         cin >> x >> y;
29         path[i] = P(x, y);
30     }
31
32     // Read the triangles
33     Trs triangles(n);
34     for (int i = 0; i < n; ++i) {
35         Pts t;
36         for (std::size_t j = 0; j < 6; ++j) {
37             P p;
38             std::cin >> p;
39             t.push_back(p);
40         }
41         // Ensure correct order for orientation tests
42         for (int j = 0; j < 6; j+=2)
43             if (CGAL::right_turn(t[j],t[j+1],t[(j+2)%6])) std::swap(t[j],t[j+1]);
44         // Store which path segments are covered
45         bool prev = contains(t,path[0]);
46         for (int j = 1; j < m; ++j) {
47             if (contains(t,path[j])) {
48                 if (prev) triangles[i].push_back(j-1);
49                 else prev = true;
50             } else
51                 prev = false;
52         }
53     }
54
55     // search for the cover by scanning through the sequence of triangles
56     Covered covered(m-1,0); // #times i,i+1 is covered
57     int uncovered = m-1; // #uncovered segments (covered[i]=0)
58     int best = n; // size of best range so far
59     for (int tb = 0, te = 0; tb != n; ) {
60         // Ensure covering
61         for (; uncovered > 0 && te != n; te++) {
62             for (Iterator j = triangles[te].begin(); j != triangles[te].end(); j++)
63                 if (++covered[*j] == 1) --uncovered;
64         }
65         if (uncovered != 0) break;
66         // Can we remove tb?
67         do {
68             for (Iterator j = triangles[tb].begin(); j != triangles[tb].end(); j++)
69                 if (--covered[*j] == 0) uncovered++;
70             while (++tb != te && uncovered == 0);
71             best = min(best, te - tb + 1);
72         }
73     }
74     cout << best << endl;
75 }
76
```

```
77 int main() {  
78     ios_base::sync_with_stdio(false);  
79     int T; cin >> T;  
80     for (int t=0; t < T; t++){  
81         hiking_maps();  
82     }  
83 }
```

## Antenna

**Keywords**— Minimum Circle

```
1
2 #include <CGAL/Exact_predicates_exact_constructions_kernel_with_sqrt.h>
3 #include <CGAL/Min_circle_2.h>
4 #include <CGAL/Min_circle_2_traits_2.h>
5 #include <iostream>
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 using namespace std;
13
14
15 double ceil_to_double(const K::FT& x) {
16     double a = ceil(CGAL::to_double(x));
17     while (a+1 >= x) a -= 1;
18     while (a < x) a += 1;
19     return a;
20 }
21
22 void compute_antenna_radius(int n) {
23     vector<K::Point_2> P(n);
24     long x, y;
25     for (int i = 0; i < n; i++) {
26         cin >> x >> y;
27         P[i] = K::Point_2(x, y);
28     }
29
30     Min_circle mc(P.begin(), P.end(), true);
31     Traits::Circle c = mc.circle();
32     cout << ceil_to_double(sqrt(c.squared_radius())) << endl;
33 }
34
35
36 int main() {
37     int nb_population; cin >> nb_population;
38
39     cout << fixed << setprecision(0);
40     while(nb_population > 0) {
41         compute_antenna_radius(nb_population);
42         cin >> nb_population;
43     }
44     return 0;
45 }
```

# Attack of the Clones

**Keywords**— Interval Scheduling

```
1  /*
2  The algorithm to solve this problem consist in using the Interval Scheduling to place the maximum
3  number of jedis along the perimeter. The problem begins when the range is continuously cyncing so
4  there is no reference point to start. The best point to use it, is the one with less jedis
   covering it (for the first two testsets there is ensure at least one segment without covering
   and on the last
5  testset there is at least one with at most 10 jedis). This segment with minimum jedis covering
6  should be found and then apply from there the Interval Scheduling algorithm for all combinations
   of
7  starting points (none segment is cutting the starting point, or picking each of the segments that
8  is cutting the starting point and take the best result).
9  */
10 #include <iostream>
11 #include <vector>
12 #include <map>
13 #include <algorithm>
14 #include <climits>
15
16 using namespace std;
17
18 void compute_jedis() {
19     long N, M; cin >> N >> M;
20
21     vector<pair<int, int> > jedis(N);
22     // vector<pair<int,int> > nn_jedis, sum_jedis;
23     map<int, int> nn_jedis;
24     map<int, int> sum_jedis;
25
26     int a, b;
27     for (int i = 0; i < N; i++) {
28         cin >> a >> b;
29         a--; b--; // Re-index to 0-based index
30         jedis[i] = make_pair(a, b);
31         nn_jedis[a]++;
32         if (b < a) nn_jedis[0]++;
33         if (b < M - 1) nn_jedis[b+1]--;
34     }
35
36     // Find the position which no interval overlaps, or the one that less intervals overlap (at
37     // most 10 by statement)
38     int prev_sum = 0;
39     int min_jedis = INT_MAX, min_jedis_pos = -1;
40     for (auto nn_jedi : nn_jedis) {
41         int pos = nn_jedi.first;
42         sum_jedis[pos] = prev_sum + nn_jedi.second;
43         prev_sum += nn_jedi.second;
44         // cout << pos << "\t" << prev_sum << endl;
45         if (prev_sum < min_jedis) {
46             min_jedis = prev_sum;
47             min_jedis_pos = pos;
48         }
49     }
50 }
51
52 // Separate the intervals between the ones that cut k_0 and the ones that not.
53 vector<pair<int,int> > jedis_cutting, jedis_not_cutting;
54 int ai, bi;
55 int k0 = min_jedis_pos;
56 for (int i = 0; i < N; i++) {
57     a = jedis[i].first;
58     b = jedis[i].second;
59     ai = (a - k0 + M) % M;
60     bi = (b - k0 + M) % M;
61
62     if (bi < ai) jedis_cutting.push_back(make_pair(bi, ai));
63     else jedis_not_cutting.push_back(make_pair(bi, ai));
64 }
65
66 // Sort by ending interval position
67 sort(jedis_not_cutting.begin(), jedis_not_cutting.end());
68
69 // Interval Scheduling for the non-cutting-min position intervals
70 int result = 0, lastb = -1;
71 for (int i = 0; i < jedis_not_cutting.size(); i++) {
72     if (jedis_not_cutting[i].second > lastb) {
73         result++;
74         lastb = jedis_not_cutting[i].first;
75     }
76 }
```

```

75     }
76 }
77
78 // Now the same algorithm of Interval Scheduling but for each of the interval that cut the min
79 // jedis position
80 int result_cutting;
81 for (int j = 0; j < jedis_cutting.size(); j++) {
82     result_cutting = 1;
83     lastb = jedis_cutting[j].first;
84     for (int i = 0; i < jedis_not_cutting.size(); i++) {
85         if (jedis_not_cutting[i].second > lastb && jedis_not_cutting[i].first < jedis_cutting[
            j].second) {
86             result_cutting++;
87             lastb = jedis_not_cutting[i].first;
88         }
89     }
90     result = max(result, result_cutting);
91 }
92
93 cout << result << endl;
94 }
95
96
97 int main() {
98     ios_base::sync_with_stdio(false);
99     int t; cin >> t;
100
101     for (int i = 0; i < t; i++) {
102         compute_jedis();
103     }
104     return 0;
105 }

```

## 1.4 BGL

### First Steps BGL

*Keywords*— Minimum Spanning Tree, Dijkstra Shortest Path

```
1 // STL includes
2 #include <vector>
3 #include <iostream>
4 #include <algorithm>
5 #include <climits>
6 // BGL includes
7 #include <boost/graph/adjacency_list.hpp>
8 #include <boost/graph/strong_components.hpp>
9 #include <boost/graph/dijkstra_shortest_paths.hpp>
10 #include <boost/graph/prim_minimum_spanning_tree.hpp>
11
12 // Namespaces
13 using namespace std;
14 using namespace boost;
15
16 // Directed graph with integer weights on edges.
17 typedef adjacency_list<vecS, vecS, undirectedS, no_property, property<edge_weight_t, int> > Graph;
18 typedef graph_traits<Graph>::vertex_descriptor Vertex;
19 typedef graph_traits<Graph>::edge_descriptor Edge;
20 typedef graph_traits<Graph>::edge_iterator EdgeIt;
21 // Property map edge -> weight
22 typedef property_map<Graph, edge_weight_t::type> WeightMap;
23
24
25 void graphs() {
26     int n, m;
27     cin >> n >> m;
28
29     // Initialize graph and weightmap
30     Graph G(n);
31     WeightMap weightmap = get(edge_weight, G);
32
33     // Store the graph
34     for (int i = 0; i < m; i++) {
35         int u, v, w;
36         cin >> u >> v >> w;
37         Edge e; bool success;
38         tie(e, success) = add_edge(v, u, G);
39         weightmap[e] = w;
40     }
41
42     // Find the Minimum Spanning Tree with Prim algorithm
43     vector<int> predmap(n);
44     Vertex start = 0;
45     prim_minimum_spanning_tree(G, make_iterator_property_map(
46         predmap.begin(), get(vertex_index, G)
47     ), root_vertex(start));
48     int w = 0;
49     for (int j = 0; j < n; j++) {
50         Edge e; bool success;
51         tie(e, success) = edge(j, predmap[j], G);
52         if (success) {
53             w += weightmap[e];
54         }
55     }
56
57     // Compute shortest t-u path in G
58     vector<int> distmap(n);
59     vector<Vertex> predmap_d(n);
60     dijkstra_shortest_paths(G, start,
61         predecessor_map(make_iterator_property_map(
62             predmap_d.begin(), get(vertex_index, G)
63         )),
64         distance_map(make_iterator_property_map(
65             distmap.begin(), get(vertex_index, G)
66         ))
67     );
68
69     // Find the furthest vertex
70     int max_distance = 0;
71     for (int j = 1; j < n; j++) {
72         if (distmap[j] < INT_MAX) {
73             max_distance = max(max_distance, distmap[j]);
74         }
75     }
```



```
76     }
77
78     cout << w << " " << max_distance << endl;
79
80 }
81
82 int main() {
83     ios_base::sync_with_stdio(false);
84     int T; cin >> T;
85     for (int t=0; t < T; t++){
86         graphs();
87     }
88 }
```

## Ant Challenge

**Keywords**— Minimum Spanning Tree, Dijkstra Shortest Path

```
1 // STL includes
2 #include <vector>
3 #include <iostream>
4 #include <algorithm>
5 #include <climits>
6 // BGL includes
7 #include <boost/graph/adjacency_list.hpp>
8 #include <boost/graph/strong_components.hpp>
9 #include <boost/graph/dijkstra_shortest_paths.hpp>
10 #include <boost/graph/prim_minimum_spanning_tree.hpp>
11 #include <boost/graph/graph_utility.hpp>
12
13 // Namespaces
14 using namespace std;
15 using namespace boost;
16
17 // Directed graph with integer weights on edges.
18 typedef adjacency_list<vecS, vecS, undirectedS, no_property, property<edge_weight_t, int> > Graph;
19 typedef graph_traits<Graph>::vertex_descriptor Vertex;
20 typedef graph_traits<Graph>::edge_descriptor Edge;
21 typedef graph_traits<Graph>::edge_iterator EdgeIt;
22 // Property map edge -> weight
23 typedef property_map<Graph, edge_weight_t>::type WeightMap;
24
25 void graphs() {
26     int n, e, s, a, b;
27     cin >> n >> e >> s >> a >> b;
28
29     vector<Graph> Gs(s);
30     vector<WeightMap> weightmaps(s);
31     for (int i = 0; i < s; i++) {
32         weightmaps[i] = get(edge_weight, Gs[i]);
33     }
34
35     for (int i = 0; i < e; i++) {
36         int u, v;
37         cin >> u >> v;
38
39         for (int j = 0; j < s; j++) {
40             int w; cin >> w;
41             Edge edge_to_add; bool success;
42             tie(edge_to_add, success) = add_edge(u, v, Gs[j]);
43             weightmaps[j][edge_to_add] = w;
44         }
45     }
46
47     Graph G;
48     WeightMap weightmap = get(edge_weight, G);
49
50     // For every specie compute the Prim Minimum Spanning Tree and then add the edges into a
51     // global
52     // graph
53     for (int i = 0; i < s; i++) {
54         int v; cin >> v;
55         vector<Vertex> primpredmap(n);
56         prim_minimum_spanning_tree(Gs[i], make_iterator_property_map(primpredmap.begin(), get(
57             vertex_index, Gs[i])));
58
59         Edge edge_s; bool success;
60         Edge new_edge; bool new_success;
61         for (int j = 0; j < n; j++) {
62             if (j == primpredmap[j]) { continue; }
63
64             Vertex source = j, target = primpredmap[j];
65             tie(edge_s, success) = edge(source, target, Gs[i]);
66             int weight = weightmaps[i][edge_s];
67             tie(new_edge, new_success) = add_edge(j, primpredmap[j], G);
68             weightmap[new_edge] = weight;
69         }
70     }
71
72     // Compute the dijkstra shortest path over the global graph
73     vector<int> distmap(n);
74     vector<Vertex> predmap(n);
75 }
```

```

76     Vertex start = a;
77     Vertex end = b;
78     dijkstra_shortest_paths(G, start,
79         predecessor_map(make_iterator_property_map(
80             predmap.begin(), get(vertex_index, G)
81         )),
82         distance_map(make_iterator_property_map(
83             distmap.begin(), get(vertex_index, G)
84         ))
85     );
86
87     cout << distmap[end] << endl;
88 }
89
90 int main() {
91     ios_base::sync_with_stdio(false);
92     int T; cin >> T;
93     for (int t=0; t < T; t++){
94         graphs();
95     }
96 }

```

## Important Bridges

*Keywords*— Biconnected Components

```
1 // STL includes
2 #include <vector>
3 #include <iostream>
4 #include <algorithm>
5 #include <climits>
6 // BGL includes
7 #include <boost/graph/adjacency_list.hpp>
8 #include <boost/graph/strong_components.hpp>
9 #include <boost/graph/dijkstra_shortest_paths.hpp>
10 #include <boost/graph/prim_minimum_spanning_tree.hpp>
11 #include <boost/graph/biconnected_components.hpp>
12
13
14
15 // Namespaces
16 using namespace std;
17 namespace boost {
18     struct edge_component_t {
19         enum { num = 555 };
20         typedef edge_property_tag kind;
21     } edge_component;
22 }
23
24 using namespace boost;
25
26 // Directed graph with integer weights on edges.
27 typedef adjacency_list<vecS, vecS, undirectedS, no_property, property <edge_component_t, size_t> >
    Graph;
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
32
33 void important_bridges() {
34     int n, m;
35     cin >> n >> m;
36
37     Graph G;
38     vector<pair<int, int> > bridges(m), critical_bridges;
39     for (int i = 0; i < m; i++) {
40         int u, v;
41         cin >> u >> v;
42         bridges[i] = pair<int, int>(u, v);
43         Edge e; bool success;
44         tie(e, success) = add_edge(u, v, G);
45     }
46
47     property_map<Graph, edge_component_t>::type
48     component = get(edge_component, G);
49
50     size_t num_comps = biconnected_components(G, component);
51
52     // Agrupate all the Edges depending the component ID
53     EdgeIt ei, ei_end;
54     vector<vector<Edge> > edge_components(num_comps, vector<Edge>(0));
55     for (tie(ei, ei_end) = edges(G); ei != ei_end; ++ei) {
56         int comp = component[*ei];
57         edge_components[comp].push_back(*ei);
58     }
59
60     // Store as result all the edges that are alone at its component
61     for (int i = 0; i < num_comps; i++) {
62         if (edge_components[i].size() > 1) {
63             continue;
64         }
65         Vertex s, t;
66         s = source(edge_components[i][0], G);
67         t = target(edge_components[i][0], G);
68         if (s < t) {
69             critical_bridges.push_back(pair<int, int>(s, t));
70         } else {
71             critical_bridges.push_back(pair<int, int>(t, s));
72         }
73     }
74
75     // Print number of critical bridges
76     sort(critical_bridges.begin(), critical_bridges.end());
```

```
77     cout << critical_bridges.size() << endl;
78     for (int i = 0; i < critical_bridges.size(); i++) {
79         cout << critical_bridges[i].first << " " << critical_bridges[i].second << endl;
80     }
81 }
82
83
84 int main() {
85     ios_base::sync_with_stdio(false);
86     int T; cin >> T;
87     for (int t=0; t < T; t++){
88         important_bridges();
89     }
90 }
```

## Buddy Selection

*Keywords*— Maximum Cardinality Matching

```
1 // STL includes
2 #include <vector>
3 #include <string>
4 #include <iostream>
5 #include <unordered_map>
6 // BGL includes
7 #include <boost/graph/adjacency_list.hpp>
8 #include <boost/graph/max_cardinality_matching.hpp>
9
10 // Namespaces
11 using namespace std;
12 using namespace boost;
13
14 // Directed graph with integer weights on edges.
15 typedef adjacency_list<vecS, vecS, undirectedS, no_property, property<edge_weight_t, int> > Graph;
16 typedef graph_traits<Graph>::vertex_descriptor Vertex;
17 typedef graph_traits<Graph>::edge_descriptor Edge;
18 typedef graph_traits<Graph>::edge_iterator EdgeIt;
19 // Property map edge -> weight
20 typedef property_map<Graph, edge_weight_t>::type WeightMap;
21
22
23
24 void buddy_selection() {
25     int n, c, f;
26     cin >> n >> c >> f;
27
28     vector<vector<int> > adjacency_list(n, vector<int>(n, 0));
29
30     /* Create the graph with all the edges that have higher weight than f and perform maximum
31     cardinality matching to find a more optimal solution of the problem. If cardinality matching
32     is
33     equal to n/2 it means that the solution proposed 'f' is not the optimal */
34     std::unordered_map<string, vector<int> > characteristics;
35     string name;
36     for (int i = 0; i < n; i++) {
37         for (int j = 0; j < c; j++) {
38             cin >> name;
39             characteristics[name].push_back(i);
40         }
41     }
42
43     // Iterate over all characteristics and create vertex with weight
44     for (auto it = characteristics.begin(); it != characteristics.end(); it++) {
45         vector<int> people = it->second;
46         int np = people.size(); // Number of people that share the same characteristic
47         if (np <= 1) continue;
48
49         // Lets iterate over all people that share the same characteristic
50         for (int i = 0; i < np; i++) {
51             for (int j = i+1; j < np; j++) {
52                 int p_i = people[i], p_j = people[j];
53                 adjacency_list[p_i][p_j]++;
54                 adjacency_list[p_j][p_i]++;
55             }
56         }
57     }
58
59     // Create final graph
60     Graph G(n);
61     WeightMap weightmap = get(edge_weight, G);
62
63     // Now iterate over all edges and subtract the edges with a weight less than 'f'
64     for (int i = 0; i < n; i++) {
65         for (int j = i + 1; j < n; j++) {
66             int weight = adjacency_list[i][j];
67             if (weight > f) {
68                 Edge edg; bool success;
69                 tie(edg, success) = add_edge(i, j, G);
70                 weightmap[edg] = weight - f;
71             }
72         }
73     }
74
75     // Compute Matching
76     vector<Vertex> matemap(n); // Use as an Exterior Property Map: Vertex -> Matc
77     edmonds_maximum_cardinality_matching (
```

```

77         G,
78         make_iterator_property_map(matemap.begin(), get(vertex_index , G))
79     );
80     // Look at the matching size
81     int matchingsize = matching_size (
82         G,
83         make_iterator_property_map (matemap.begin(), get(vertex_index , G))
84     );
85
86     if (matchingsize == n/2) {
87         cout << "not optimal" << endl;
88     } else {
89         cout << "optimal" << endl;
90     }
91 }
92
93 int main() {
94     ios_base::sync_with_stdio(false);
95     int T; cin >> T;
96     for (int t=0; t < T; t++){
97         buddy_selection();
98     }
99 }

```

```

1
2 #include <CGAL/Exact_predicates_exact_constructions_kernel.h>
3 #include <CGAL/Min_circle_2.h>
4 #include <CGAL/Min_circle_2_traits_2.h>
5 #include <iostream>
6 #include <vector>
7
8 typedef CGAL::Exact_predicates_exact_constructions_kernel K;
9 typedef K::Point_2 P;
10 typedef CGAL::Min_circle_2_traits_2<K> Traits;
11 typedef CGAL::Min_circle_2<Traits> Min_circle;
12
13 using namespace std;
14
15
16 double ceil_to_double(const K::FT& x) {
17     double a = ceil(CGAL::to_double(x));
18     while (a + 1 >= x) a -= 1;
19     while (a < x) a += 1;
20     return a;
21 }
22
23
24 pair<bool, K::FT> evaluate(int index, vector<K::FT>& distances, vector<P>& cities) {
25
26     // Initialize the external circle with the furthest cities from 0 to index.
27     Min_circle min_ex_circle(cities.begin(), cities.begin() + index + 1, true);
28     Traits::Circle circle = min_ex_circle.circle();
29
30     K::FT ex_radius = circle.squared_radius();
31     K::FT tc_radius;
32     if (index >= cities.size() - 1) tc_radius = 0;
33     else tc_radius = distances[index + 1];
34
35     bool eval = ex_radius >= tc_radius;
36     K::FT max_radius = max(ex_radius, tc_radius);
37
38     return make_pair(eval, max_radius);
39 }
40
41 int binary_search(int left, int right, vector<K::FT>& distances, vector<P>& cities) {
42     if (left == right - 1) return left;
43     int m = (right + left) / 2;
44
45     K::FT max_rad; bool eval;
46     tie(eval, max_rad) = evaluate(m, distances, cities);
47     if (!eval) {
48         return binary_search(m, right, distances, cities);
49     } else {
50         return binary_search(left, m, distances, cities);
51     }
52 }
53
54 void theev() {
55     int n; cin >> n;
56
57     // Read cities locations
58     long x, y; cin >> x >> y;
59     P their_city(x, y);
60     P e_their_city(x, y);
61
62     vector<P> cities(n-1);
63     for (int i = 0; i < n-1; i++) {
64         cin >> x >> y;
65         cities[i] = P(x, y);
66     }
67
68     // For the cas we have one or two cities where the min radius will be 0
69     if (n <= 2) {
70         cout << 0 << endl;
71         return;
72     }
73
74     // Sort all the cities depending on the distance to the capital with decreasing order
75     sort(cities.begin(), cities.end(),
76         [&their_city](P a, P b) {
77             return CGAL::squared_distance(a, their_city) > CGAL::squared_distance(b, their_city);

```



```

78     }
79 );
80
81 // Compute and store the distances to the capital
82 vector<K::FT> distances(n-1);
83 for (int i = 0; i < n - 1; i++) {
84     distances[i] = squared_distance(cities[i], their_city);
85 }
86
87 // Perform binary search
88 int index = binary_search(0, n-1, distances, cities);
89
90 K::FT rad_1 = evaluate(index, distances, cities).second;
91 K::FT rad_2 = evaluate(index + 1, distances, cities).second;
92
93 K::FT result = min(rad_1, rad_2);
94
95 cout << ceil_to_double(result) << endl;
96 }
97
98 int main() {
99     int t; cin >> t;
100
101     cout << fixed << setprecision(0);
102     for (int i = 0; i < t; i++) {
103         theev();
104     }
105     return 0;
106 }

```

## Chapter 2

# Advanced Algorithms

## 2.1 Dynamic Programming, Brute Force, Split and List

### Burning Coins

*Keywords*— Dynamic Programming

```
1 #include <vector>
2 #include <iostream>
3
4 using namespace std;
5
6
7 int play(vector<int>& coins_values, int left_idx, int right_idx, bool turn, vector<vector<int> >&
   mem) {
8     /* turn = true -> Your turn
9        turn = false -> Friend's turn */
10
11     // Final case if no coins left
12     if (left_idx == right_idx) {
13         return 0;
14     }
15
16     if (mem[left_idx][right_idx-1] > -1) {
17         return mem[left_idx][right_idx-1];
18     }
19
20     // First take the left coins then the right one
21     int left_result, right_result, result;
22     if (turn) {
23         left_result = coins_values[left_idx];
24         left_result += play(coins_values, left_idx+1, right_idx, false, mem);
25         right_result = coins_values[right_idx-1];
26         right_result += play(coins_values, left_idx, right_idx-1, false, mem);
27         result = (left_result > right_result) ? left_result : right_result;
28         mem[left_idx][right_idx-1] = result;
29         return result;
30     } else {
31         left_result = play(coins_values, left_idx+1, right_idx, true, mem);
32         right_result = play(coins_values, left_idx, right_idx-1, true, mem);
33         result = (left_result < right_result) ? left_result : right_result;
34         mem[left_idx][right_idx-1] = result;
35         return result;
36     }
37 }
38
39
40 void burning_coins() {
41     int n; cin >> n;
42
43     vector<int> coins_values(n);
44     for (int i = 0; i < n; i++) {
45         cin >> coins_values[i];
46     }
47
48     // Add some memory storing the result for every pair of left and right indexes possible (
       speedup)
49     vector<vector<int> > mem(n, vector<int>(n, -1));
50     cout << play(coins_values, 0, n, true, mem) << endl;
51
52
53 }
54
```

```
55 int main() {  
56     ios_base::sync_with_stdio(false);  
57     int T; cin >> T;  
58     for (int t=0; t < T; t++){  
59         burning_coins();  
60     }  
61 }
```

# Light Pattern

**Keywords**— Dynamic Programming

```
1  #include <vector>
2  #include <iostream>
3
4  using namespace std;
5
6
7  void light_pattern() {
8      int n, k, x;
9      cin >> n >> k >> x;
10
11
12      int p, s, b;          // p: number of patterns,
13      uint16_t q = 0x00;    // q: desired pattern
14      p = n / k;
15
16      // Compute the desired pattern
17      for (int i = 1; i <= k; i++) {
18          q |= (x & 1) << (k-i);
19          x >>= 1;
20      }
21
22      vector<uint16_t> bulbs_state(p);
23      for (int i = 0; i < p; i++) {
24          s = 0;              // s to store the bulb state
25          for (int j = 0; j < k; j++) {
26              cin >> b;
27              if (b == 1) s |= 1 << j;
28          }
29          bulbs_state[i] = s;
30      }
31
32      // Compute the diff state of each pattern with the desired one. The int represent the number
33      // of different bulbs at each of the patterns.
34      vector<int> diff_state(p, 0);
35      for (int i = 0; i < p; i++) {
36          int diff = bulbs_state[i] ^ q;
37          for (int j = 0; j < k; j++) {
38              if ((diff & 1 << j) != 0) diff_state[i]++;
39          }
40      }
41
42      // Compute the accumulated in case the bulbs have been changed one by one or by block all the
43      // previous ones
44      vector<int> values_individual(p), values_block(p);
45      values_individual[0] = diff_state[0];
46      values_block[0] = k - diff_state[0] + 1;
47      for (int i = 1; i < p; i++) {
48          values_individual[i] = min(values_individual[i-1], values_block[i-1]) + diff_state[i];
49          values_block[i] = min(values_block[i-1] + k - diff_state[i],
50                               values_individual[i-1] + k - diff_state[i] + 2);
51      }
52
53      cout << min(values_individual[p-1], values_block[p-1]) << endl;
54  }
55
56  int main() {
57      ios_base::sync_with_stdio(false);
58      int T; cin >> T;
59      for (int t=0; t < T; t++){
60          light_pattern();
61      }
62  }
```

## Light at the Museum

**Keywords**— Dynamic Programming, Split and List

```
1 #include <vector>
2 #include <iostream>
3 #include <algorithm>
4 #include <climits>
5
6 using namespace std;
7
8 typedef pair<int, int> pair_int;
9 typedef vector<vector<int>> > vecvec;
10 typedef vector<pair<vector<int>,int> > vecpairvec;
11
12
13 void create_subset(vecvec& state_on, vecvec& state_off, vecpairvec& F, int dbound, int ubound) {
14     int N = ubound - dbound;
15     int M = state_on[0].size();
16     for (int k = 0; k < 1<<N; k++) {
17         vector<int> tuple(M, 0);
18         int count = 0;
19
20         for (int i = 0; i < N; i++) {
21             bool changed_state = (k & 1<<i) != 0;
22             if (changed_state) count++;
23
24             for (int j = 0; j < M; j++)
25                 tuple[j] += (changed_state) ? state_off[i+dbound][j] : state_on[i+dbound][j];
26         }
27         F.push_back(make_pair(tuple, count));
28     }
29 }
30
31 struct Comp {
32     bool operator() (pair<vector<int>, int> P, vector<int> V) {
33         return P.first < V;
34     }
35     bool operator() (vector<int> V, pair<vector<int>, int> P) {
36         return V < P.first;
37     }
38 };
39
40 void light_at_the_museum() {
41     int N, M; cin >> N >> M;
42
43     vector<int> brightness(M);
44     for (int i = 0; i < M; i++) cin >> brightness[i];
45
46     vector<vector<int>> state_on(N, vector<int>(M)), state_off(N, vector<int>(M));
47     for (int j = 0; j < N; j++)
48         for (int i = 0; i < M; i++) cin >> state_on[j][i] >> state_off[j][i];
49
50     // Create all combinations for two subsets (divide and list)
51     vecpairvec F1, F2;
52     create_subset(state_on, state_off, F1, 0, N/2);
53     create_subset(state_on, state_off, F2, N/2, N);
54     sort(F2.begin(), F2.end());
55
56     int min_changes = INT_MAX;
57     for (int idx = 0; idx < F1.size(); idx++) {
58         vector<int> missing = F1[idx].first;
59         for (int i = 0; i < M; i++) {
60             missing[i] = brightness[i] - missing[i];
61         }
62
63         pair<vecpairvec::iterator, vecpairvec::iterator> it_pair;
64         it_pair = equal_range(F2.begin(), F2.end(), missing, Comp());
65
66         for (auto it = it_pair.first; it != it_pair.second; it++) {
67             int count = it->second + F1[idx].second;
68             min_changes = min(min_changes, count);
69         }
70     }
71
72     if (min_changes == INT_MAX) cout << "impossible\n";
73     else cout << min_changes << endl;
74 }
75
76 int main() {
```

```
78 ios_base::sync_with_stdio(false);
79 int T; cin >> T;
80 for (int t=0; t < T; t++){
81     light_at_the_museum();
82 }
83 }
```

# The Great Game

*Keywords*— Dynamic Programming

```
1  #include <vector>
2  #include <iostream>
3  #include <climits>
4
5  using namespace std;
6
7
8  int max_(vector<vector<int> > &transitions, int start, int end,
9          vector<int> &min_mem, vector<int> &max_mem);
10
11 int min_(vector<vector<int> > &transitions, int start, int end,
12          vector<int> &min_mem, vector<int> &max_mem) {
13
14     if (start == end) {
15         min_mem[start] = 0;
16         return 0;
17     }
18
19     int min_dist = INT_MAX;
20     for (int v : transitions[start]) {
21         if (max_mem[v] == -1) {
22             max_(transitions, v, end, min_mem, max_mem);
23         }
24         min_dist = min(min_dist, max_mem[v]);
25     }
26     min_mem[start] = min_dist + 1;
27     return min_mem[start];
28 }
29
30
31 int max_(vector<vector<int> > &transitions, int start, int end,
32          vector<int> &min_mem, vector<int> &max_mem) {
33
34     if (start == end) {
35         max_mem[start] = 0;
36         return 0;
37     }
38
39     int max_dist = 0;
40     for (int v : transitions[start]) {
41         if (min_mem[v] == -1) {
42             min_(transitions, v, end, min_mem, max_mem);
43         }
44         max_dist = max(max_dist, min_mem[v]);
45     }
46     max_mem[start] = max_dist + 1;
47     return max_mem[start];
48 }
49
50 void great_game() {
51     int n, m;
52     cin >> n >> m;
53     int r, b;
54     cin >> r >> b;
55     // set to 0-index
56     r--; b--;
57
58     vector<vector<int> > transitions(n);
59     for (int i = 0; i < m; i++) {
60         int u, v;
61         cin >> u >> v;
62         // set to 0-index
63         u--; v--;
64         transitions[u].push_back(v);
65     }
66
67     // Create memory vectors
68     vector<int> min_mem(n, -1), max_mem(n, -1);
69
70     // For each of the starting possitions, check which one has minimum movements to win the game.
71     int red_moves = min_(transitions, r, n-1, min_mem, max_mem);
72     int black_moves = min_(transitions, b, n-1, min_mem, max_mem);
73
74     cout << (black_moves < red_moves || (black_moves == red_moves && black_moves % 2 == 0)) <<
75         endl;
76 }
```

```
77 int main() {  
78     ios_base::sync_with_stdio(false);  
79     int T; cin >> T;  
80     for (int t=0; t < T; t++){  
81         great_game();  
82     }  
83 }
```



# On Her Majesty's Secret Service

**Keywords**— Maximum Cardinality Matching, Dijkstra Shortest Path

```
1 // STL includes
2 #include <vector>
3 #include <string>
4 #include <iostream>
5 #include <unordered_map>
6 // BGL includes
7 #include <boost/graph/adjacency_list.hpp>
8 #include <boost/graph/max_cardinality_matching.hpp>
9 #include <boost/graph/dijkstra_shortest_paths.hpp>
10
11 // Namespaces
12 using namespace std;
13 using namespace boost;
14
15 // Directed graph with integer weights on edges.
16 typedef adjacency_list<vecS, vecS, undirectedS, no_property, property<edge_weight_t, int> > Graph;
17 typedef adjacency_list<vecS, vecS, directedS, no_property, property<edge_weight_t, int> > DiGraph;
18 typedef graph_traits<DiGraph>::vertex_descriptor Vertex;
19 typedef graph_traits<DiGraph>::edge_descriptor Edge;
20 typedef graph_traits<DiGraph>::edge_iterator EdgeIt;
21 // Property map edge -> weight
22 typedef property_map<DiGraph, edge_weight_t>::type WeightMap;
23
24
25 void oh_her_majestys_secret_service() {
26     int n, m, a, s, c, d;
27     cin >> n >> m >> a >> s >> c >> d;
28
29     DiGraph G(n);
30     WeightMap weightmap = get(edge_weight, G);
31
32     char w; int x, y, z;
33     Edge edg; bool success;
34     for (int i = 0; i < m; i++) {
35         cin >> w >> x >> y >> z;
36
37         tie(edg, success) = add_edge(x, y, G);
38         weightmap[edg] = z;
39         if (w == 'L') {
40             tie(edg, success) = add_edge(y, x, G);
41             weightmap[edg] = z;
42         }
43     }
44 }
45
46 vector<int> agents(a), shelters(s);
47 for (int i = 0; i < a; i++) cin >> agents[i];
48 for (int i = 0; i < s; i++) cin >> shelters[i];
49
50 // Compute one distance map per agent
51 vector<vector<int> > distmap(a, vector<int>(n));
52 for (int i = 0; i < a; i++) {
53     dijkstra_shortest_paths(G, agents[i],
54         distance_map(make_iterator_property_map(distmap[i].begin(), get(vertex_index, G))));
55 }
56 // Represent G' as the pairwise distance matrix T from agents to shelters.
57 vector<vector<int> > T(a, vector<int>(s, INT_MAX));
58 for (int i = 0; i < a; i++) {
59     for (int j = 0; j < s; j++) {
60         T[i][j] = distmap[i][shelters[j]];
61     }
62 }
63
64 // Binary search for the smallest t
65 int low = 0, high = INT_MAX;
66 while (low < high) {
67     int mid = low + (high-low)/2;
68     // Represent the model as a bipartite graph with all the agents as nodes in one side, and
69     // all the shelters duplicated for each possible capacity and then compute the maximum
70     // cardinality matching
71     Graph GG(a + s*c);
72     for (int i = 0; i < a; i++) {
73         for (int j = 0; j < s; j++) {
74             if (T[i][j] == INT_MAX) continue;
75             for (int k = 0; k < c; k++) {
76                 if (T[i][j] + (k + 1) * d <= mid) {
77                     add_edge(i, a + k * s + j, GG);
78                 }
79             }
80         }
81     }
82 }
```

```

78         }
79     }
80 }
81 }
82 // Compute maximum cardinality
83 vector<Vertex> matemap(a + s * c); // Use as an Exterior Property Map: Vertex -> Matc
84 edmonds_maximum_cardinality_matching (
85     GG,
86     make_iterator_property_map(matemap.begin(), get(vertex_index , G))
87 );
88 const Vertex NULL_VERTEX = graph_traits<Graph>::null_vertex();
89 int matchingsize = 0;
90 for (int i = 0; i < a; i++) {
91     matchingsize += (matemap[i] != NULL_VERTEX);
92 }
93
94 if (matchingsize == a) high = mid;
95 else low = mid + 1;
96
97 }
98 cout << low << endl;
99 }
100
101 int main() {
102     ios_base::sync_with_stdio(false);
103     int T; cin >> T;
104     for (int t=0; t < T; t++){
105         oh_her_majestys_secret_service();
106     }
107 }

```

# Poker Chips

*Keywords*— Dynamic Programming

```
1  #include <vector>
2  #include <map>
3  #include <iostream>
4  #include <cmath>
5  #include <stdexcept>
6  #include <algorithm>
7
8  using namespace std;
9
10 double compute_award(double k) {
11     if (k > 1) {
12         return exp2(k-2);
13     } else if (k == 1) {
14         return 0;
15     } else {
16         throw runtime_error("k can not be less than 1.");
17     }
18 }
19
20 int map_positions_into_vector(vector<int>& stack_heigh, vector<int>& stack_top) {
21     int nb_stacks = stack_heigh.size();
22
23     int accum = 1, pos = 0;
24     for (int i = nb_stacks - 1; i >= 0; i--) {
25         pos += accum * stack_heigh[i];
26         accum *= stack_top[i] + 1;
27     }
28     return pos;
29 }
30
31 int play(vector<vector<int>> & stacks, vector<int>& nb_chips, vector<int>& turn_positions, vector<
int>& memory) {
32
33     // Check memory first
34     int mem_pos = map_positions_into_vector(turn_positions, nb_chips);
35     if (memory[mem_pos] != -1) return memory[mem_pos];
36
37     // Check if there is not more chips at the stack (it is not necessary as memory[0] = 0)
38
39     int nb_stacks = stacks.size();
40     map<int, vector<int>> > color_map;
41     // Check at each of the stacks and see the next poker chips and group them by color
42     for (int i = 0; i < nb_stacks; i++) {
43         int next_position = turn_positions[i] - 1;
44         if (next_position < 0) continue; // In case a stack is empty
45         color_map[stacks[i][next_position]].push_back(i);
46     }
47
48     // Iterate over all groups of colors, and compute the maximum award in DP achievable from this
49     // stack positions
50     int max_award = 0;
51     for (auto it = color_map.begin(); it != color_map.end(); it++) {
52         vector<int> chips_positions = it->second;
53         int nb = chips_positions.size();
54
55         /* Now compute all the possible combinations of extracting these chips (extract all the
56         ones with the same colos is not allways the best strategy)
57         Example:
58         3 3 3 2 2 2 2
59         2 2 2 X X 3 X
60
61         If use the strategy to extract all the chips from the same color award = 10
62         but if first is removed the 2 with a 3 at the bottom, then the award = 20
63         */
64         for (int k = 1; k < 1<<nb; k++) {
65             int count = 0;
66
67             // Selecting the chips from the same color, see which is the award with the next
68             position
69             vector<int> new_positions(turn_positions);
70
71             for (int i = 0; i < nb; i++) {
72                 if (k & (1<<i)) {
73                     count++;
74                     new_positions[chips_positions[i]]--;
75                 }
76             }
77         }
78     }
79 }
```

```

76         int new_award = compute_award(count);
77
78         int award = play(stacks, nb_chips, new_positions, memory);
79         max_award = max(max_award, award + new_award);
80     }
81 }
82
83 // Store result into memory
84 memory[mem_pos] = max_award;
85
86 return max_award;
87 }
88
89 void poker_chips() {
90     int n; cin >> n;
91
92     vector<int> nb_chips(n);
93     vector<vector<int>> > stacks(n);
94     vector<int> initial_positions(n);
95
96     for (int i = 0; i < n; i++) {
97         cin >> nb_chips[i];
98     }
99     for (int i = 0; i < n; i++) {
100         for (int j = 0; j < nb_chips[i]; j++) {
101             int c; cin >> c;
102             stacks[i].push_back(c);
103         }
104         initial_positions[i] = nb_chips[i];
105     }
106
107     int N_memory = map_positions_into_vector(nb_chips, nb_chips) + 1;
108     vector<int> memory(N_memory, -1);
109     memory[0] = 0;
110
111     int max_award = play(stacks, nb_chips, initial_positions, memory);
112     cout << max_award << endl;
113 }
114
115 int main() {
116     ios_base::sync_with_stdio(false);
117     int T; cin >> T;
118     for (int t=0; t < T; t++){
119         poker_chips();
120     }
121 }
122
123

```

## 2.2 BGL Flows

### Coin Tossing

*Keywords*— Maximum Flow

```
1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4  // BGL includes
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  // Namespaces
9  using namespace std;
10 using namespace boost;
11
12
13
14 // BGL Graph definitions
15 // =====
16 // Graph Type with nested interior edge properties for Flow Algorithms
17 typedef adjacency_list_traits<vecS, vecS, directedS> Traits;
18 typedef adjacency_list<vecS, vecS, directedS, no_property,
19     property<edge_capacity_t, long,
20     property<edge_residual_capacity_t, long,
21     property<edge_reverse_t, Traits::edge_descriptor> > > > Graph;
22 // Interior Property Maps
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
29
30 // Custom Edge Adder Class, that holds the references
31 // to the graph, capacity map and reverse edge map
32 // =====
33 class EdgeAdder {
34     Graph &G;
35     EdgeCapacityMap &capacitymap;
36     ReverseEdgeMap &revedgemap;
37
38 public:
39     // to initialize the Object
40     EdgeAdder(Graph & G, EdgeCapacityMap &capacitymap, ReverseEdgeMap &revedgemap):
41         G(G), capacitymap(capacitymap), revedgemap(revedgemap){}
42
43     // to use the Function (add an edge)
44     void addEdge(int from, int to, long capacity) {
45         Edge e, reverseE;
46         bool success;
47         tie(e, success) = add_edge(from, to, G);
48         tie(reverseE, success) = add_edge(to, from, G);
49         capacitymap[e] = capacity;
50         capacitymap[reverseE] = 0;
51         revedgemap[e] = reverseE;
52         revedgemap[reverseE] = e;
53     }
54 };
55
56
57 void coin_tossing() {
58     int n, m;
59     cin >> n >> m;
60
61     Graph G;
62     EdgeCapacityMap capacitymap = get(edge_capacity, G);
63     ReverseEdgeMap revedgemap = get(edge_reverse, G);
64     ResidualCapacityMap rescapacitymap = get(edge_residual_capacity, G);
65     EdgeAdder eaG(G, capacitymap, revedgemap);
66
67     for (int i = 0; i < m; i++) {
68         int a, b, c;
69         cin >> a >> b >> c;
70
71         if (c == 1) {
72             eaG.addEdge(i, m+a, 1);
73         } else if (c == 2) {
74             eaG.addEdge(i, m+b, 1);
75         } else {
```

```

76         eaG.addEdge(i, m+a, 1);
77         eaG.addEdge(i, m+b, 1);
78     }
79
80 }
81
82 Vertex src = add_vertex(G);
83 for (int i = 0; i < m; i++) {
84     eaG.addEdge(src, i, 1);
85 }
86 Vertex sink = add_vertex(G);
87 long sum = 0;
88 for (int j = m; j < m+n; j++) {
89     int s; cin >> s;
90     sum += s;
91     eaG.addEdge(j, sink, s);
92 }
93
94 long flow = push_relabel_max_flow(G, src, sink);
95
96 if (flow != m || flow != sum) {
97     cout << "no" << endl;
98 } else {
99     cout << "yes" << endl;
100 }
101 }
102
103 int main() {
104     ios_base::sync_with_stdio(false);
105     int T; cin >> T;
106     for (int t=0; t < T; t++){
107         coin_tossing();
108     }
109 }

```

# Shopping Trip

*Keywords*— Maximum Flow

```
1
2 #include <iostream>
3 #include <vector>
4 #include <algorithm>
5 // BGL includes
6 #include <boost/graph/adjacency_list.hpp>
7 #include <boost/graph/push_relabel_max_flow.hpp>
8 // Namespaces
9 using namespace std;
10 using namespace boost;
11
12
13 // BGL Graph definitions
14 // =====
15 // Graph Type with nested interior edge properties for Flow Algorithms
16 typedef adjacency_list_traits<vecS, vecS, directedS> Traits;
17 typedef adjacency_list<vecS, vecS, directedS, no_property,
18     property<edge_capacity_t, long,
19     property<edge_residual_capacity_t, long,
20     property<edge_reverse_t, Traits::edge_descriptor> > > > Graph;
21 // Interior Property Maps
22 typedef property_map<Graph, edge_capacity_t>::type EdgeCapacityMap;
23 typedef property_map<Graph, edge_residual_capacity_t>::type ResidualCapacityMap;
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;
27
28
29 // Custom Edge Adder Class, that holds the references
30 // to the graph, capacity map and reverse edge map
31 // =====
32 class EdgeAdder {
33     Graph &G;
34     EdgeCapacityMap &capacitymap;
35     ReverseEdgeMap &revedgemap;
36
37 public:
38     // to initialize the Object
39     EdgeAdder(Graph & G, EdgeCapacityMap &capacitymap, ReverseEdgeMap &revedgemap):
40         G(G), capacitymap(capacitymap), revedgemap(revedgemap){}
41
42     // to use the Function (add an edge)
43     void addEdge(int from, int to, long capacity) {
44         Edge e, reverseE;
45         bool success;
46         tie(e, success) = add_edge(from, to, G);
47         tie(reverseE, success) = add_edge(to, from, G);
48         capacitymap[e] = capacity;
49         capacitymap[reverseE] = 0;
50         revedgemap[e] = reverseE;
51         revedgemap[reverseE] = e;
52     }
53 };
54
55
56 void shopping() {
57     int n, m, s;
58     cin >> n >> m >> s;
59
60     Graph G;
61     EdgeCapacityMap capacitymap = get(edge_capacity, G);
62     ReverseEdgeMap revedgemap = get(edge_reverse, G);
63     EdgeAdder eaG(G, capacitymap, revedgemap);
64
65     vector<int> stores_locations(s);
66     for (int i = 0; i < s; i++) {
67         cin >> stores_locations[i];
68     }
69
70     // Add street edges
71     for (int i = 0; i < m; i++) {
72         int a, b;
73         cin >> a >> b;
74         eaG.addEdge(a, b, 1);
75         eaG.addEdge(b, a, 1);
76     }
77 }
```

```

78 // Add edges from stores to sink
79 // Vertex src = add_vertex(G);
80 Vertex sink = add_vertex(G);
81 // eaG.addEdge(src, 0, s);
82 for (int i = 0; i < s; i++) {
83     eaG.addEdge(stores_locations[i], sink, 1);
84 }
85
86 long flow = push_relabel_max_flow(G, 0, sink);
87
88 if (flow == s) {
89     cout << "yes" << endl;
90 } else {
91     cout << "no" << endl;
92 }
93
94 }
95
96 int main() {
97     ios_base::sync_with_stdio(false);
98     int T; cin >> T;
99     for (int t=0; t < T; t++){
100         shopping();
101     }
102 }

```



# Kingdom Defence

*Keywords*— Maximum Flow

```
1
2 #include <iostream>
3 #include <map>
4 #include <algorithm>
5 // BGL includes
6 #include <boost/graph/adjacency_list.hpp>
7 #include <boost/graph/push_relabel_max_flow.hpp>
8 // Namespaces
9 using namespace std;
10 using namespace boost;
11
12
13 // BGL Graph definitions
14 // =====
15 // Graph Type with nested interior edge properties for Flow Algorithms
16 typedef adjacency_list_traits<vecS, vecS, directedS> Traits;
17 typedef adjacency_list<vecS, vecS, directedS, no_property,
18     property<edge_capacity_t, long,
19     property<edge_residual_capacity_t, long,
20     property<edge_reverse_t, Traits::edge_descriptor> > > > Graph;
21 // Interior Property Maps
22 typedef property_map<Graph, edge_capacity_t>::type EdgeCapacityMap;
23 typedef property_map<Graph, edge_residual_capacity_t>::type ResidualCapacityMap;
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;
27 typedef graph_traits<Graph>::edge_iterator EdgeIt;
28
29
30 // Custom Edge Adder Class, that holds the references
31 // to the graph, capacity map and reverse edge map
32 // =====
33 class EdgeAdder {
34     Graph &G;
35     EdgeCapacityMap &capacitymap;
36     ReverseEdgeMap &revedgemap;
37
38 public:
39     // to initialize the Object
40     EdgeAdder(Graph & G, EdgeCapacityMap &capacitymap, ReverseEdgeMap &revedgemap):
41         G(G), capacitymap(capacitymap), revedgemap(revedgemap){}
42
43     // to use the Function (add an edge)
44     Edge addEdge(int from, int to, long capacity) {
45         Edge e, reverseE;
46         bool success;
47         tie(e, success) = add_edge(from, to, G);
48         tie(reverseE, success) = add_edge(to, from, G);
49         capacitymap[e] = capacity;
50         capacitymap[reverseE] = 0;
51         revedgemap[e] = reverseE;
52         revedgemap[reverseE] = e;
53         return e;
54     }
55 };
56
57
58 void kingdom_defence() {
59     int l, p;
60     cin >> l >> p;
61
62     Graph G(l+2);
63     EdgeCapacityMap capacitymap = get(edge_capacity, G);
64     ReverseEdgeMap revedgemap = get(edge_reverse, G);
65     // ResidualCapacityMap rescapacitymap = get(edge_residual_capacity, G);
66     EdgeAdder eaG(G, capacitymap, revedgemap);
67
68     Vertex src = vertex(l, G);
69     Vertex sink = vertex(l+1, G);
70
71     long total_defenders = 0;
72     long total_available = 0;
73     for (int i = 0; i < l; i++) {
74         long g, d;
75         cin >> g >> d;
76         Vertex location = vertex(i, G);
77         eaG.addEdge(src, location, g);
```

```

78     eaG.addEdge(location, sink, d);
79     total_defenders += d;
80     total_available += g;
81 }
82
83 for (int i = 0; i < p; i++) {
84     int f, t;
85     long c, C;
86     cin >> f >> t >> c >> C;
87     eaG.addEdge(f, t, C - c);
88     if (c > 0) {
89         eaG.addEdge(f, sink, c);
90         total_defenders += c;
91         eaG.addEdge(src, t, c);
92         total_available += c;
93     }
94 }
95
96 long flow = push_relabel_max_flow(G, src, sink);
97
98 bool result = flow == total_defenders;
99
100 if (result) {
101     cout << "yes" << endl;
102 } else {
103     cout << "no" << endl;
104 }
105
106 }
107
108 int main() {
109     ios_base::sync_with_stdio(false);
110     int T; cin >> T;
111     for (int t=0; t < T; t++){
112         kingdom_defence();
113     }
114 }

```

# A New Hope

*Keywords*— Dynamic Programming

```
1
2 #include <iostream>
3 #include <vector>
4 #include <map>
5 #include <algorithm>
6
7 using namespace std;
8
9 struct center {
10     center (int n) {
11         sup = vector<uint16_t>(n, 0);
12     }
13     vector<uint16_t> sup;
14     map<uint16_t, map<int, int> > extsup;
15     map<int, int> bestmap;
16 };
17
18
19 int best_stormtroopers(vector<center>& centers, int c, int mask, const int S) {
20     auto bestit = centers[c].bestmap.find(mask);
21     if (bestit != centers[c].bestmap.end())
22         return bestit->second;
23
24     int best = 0;
25     for (int k = 0; k < (0x1<<S); k++) {
26         // Mask refers to the stormtroopers that can not be choosen because are being supervised
27         if ((k & mask) != 0) continue;
28
29         bool valid = true;
30         // Check the validity looking if any of the stormtroopers supervise each other in this
31         // configurations
32         for (int i = 0; i < S && valid; i++) {
33             if (((k & (0x1<<i)) != 0) && ((k & centers[c].sup[i]) != 0)) valid = false;
34         }
35         if (!valid) continue;
36
37         int count = 0;
38         // Count the strompers being sabotaged
39         for (int i = 0; i < S; i++) {
40             if (k & (0x1<<i)) count++;
41         }
42         // Now compute the mask (the stormtroopers that on the children centers are being
43         // supervised by this configuration) and with DP count the maximum number of stormtroopers
44         // that are not being supervised by each other.
45         for (auto& esup : centers[c].extsup) {
46             int nmask = 0;
47             for (auto& p : esup.second) {
48                 if (k & 0x1<<p.first) nmask |= p.second;
49             }
50             count += best_stormtroopers(centers, esup.first, nmask, S);
51         }
52         best = max(best, count);
53     }
54     centers[c].bestmap[mask] = best;
55     return best;
56 }
57
58 void a_new_hope() {
59     int K, S, M;
60     cin >> K >> S >> M;
61
62     vector<center> centers(K, center(S));
63
64     int u, v, h, x, y;
65     for (int i = 0; i < M; i++) {
66         cin >> u >> v >> h;
67         if (u == v) {
68             for (int j = 0; j < h; j++) {
69                 cin >> x >> y;
70                 centers[u].sup[x] |= 0x1 << y;
71             }
72         }
73         else {
74             map<int, int> esup;
75             for (int j = 0; j < h; j++) {
76                 cin >> x >> y;
77                 esup[x] |= 0x1 << y;
```

```

78         }
79         centers[u].extsup[v] = esup;
80     }
81 }
82
83     cout << best_stormtroopers(centers, 0, 0, S) << endl;
84 }
85
86
87 int main() {
88     ios_base::sync_with_stdio(false);
89     int T; cin >> T;
90     for (int t=0; t < T; t++){
91         a_new_hope();
92     }
93 }

```

## 2.3 Linear/Quadratic Programing

What is the maximum?

*Keywords*— LP/QP

```
1 #include <iostream>
2 #include <cassert>
3 #include <cmath>
4 #include <CGAL/basic.h>
5 #include <CGAL/QP_models.h>
6 #include <CGAL/QP_functions.h>
7 #include <CGAL/Gmpz.h>
8
9 // choose exact integral type
10 typedef CGAL::Gmpz ET;
11
12 // program and solution types
13 typedef CGAL::Quadratic_program<long> Program;
14 typedef CGAL::Quadratic_program_solution<ET> Solution;
15
16 using namespace std;
17
18 void solve_1(int a, int b){
19     Program qp (CGAL::SMALLER, true, 0, false, 0);
20     const int X = 0;
21     const int Y = 1;
22     qp.set_a(X, 0, 1); qp.set_a(Y, 0, 1); qp.set_b(0, 4);
23     qp.set_a(X, 1, 4); qp.set_a(Y, 1, 2); qp.set_b(1, a*b);
24     qp.set_a(X, 2, -1); qp.set_a(Y, 2, 1); qp.set_b(2, 1);
25     qp.set_c(Y, -b);
26     qp.set_d(X, X, 2*a);
27
28     Solution s = CGAL::solve_quadratic_program(qp, ET());
29     assert (s.solves_quadratic_program(qp));
30     // cout << s << endl;
31
32     if (s.is_infeasible()) {
33         cout << "no" << endl;
34     } else if (s.is_unbounded()) {
35         cout << "unbounded" << endl;
36     } else {
37         assert (s.is_optimal());
38         long result = floor(-CGAL::to_double(s.objective_value()));
39         cout << result << endl;
40     }
41 }
42
43 void solve_2(int a, int b){
44     Program qp (CGAL::LARGER, false, 0, true, 0);
45     const int X = 0;
46     const int Y = 1;
47     const int Z2 = 2;
48     qp.set_a(X, 0, 1); qp.set_a(Y, 0, 1); qp.set_b(0, -4);
49     qp.set_a(X, 1, 4); qp.set_a(Y, 1, 2); qp.set_a(Z2, 1, 1); qp.set_b(1, -a*b);
50     qp.set_a(X, 2, -1); qp.set_a(Y, 2, 1); qp.set_b(2, -1);
51     qp.set_u(Z2, false, 0);
52     qp.set_d(X, X, 2*a);
53     qp.set_c(Y, b);
54     qp.set_d(Z2, Z2, 2*1);
55
56     Solution s = CGAL::solve_quadratic_program(qp, ET());
57     assert (s.solves_quadratic_program(qp));
58     // cout << s << endl;
59
60     if (s.is_infeasible()) {
61         cout << "no" << endl;
62     } else if (s.is_unbounded()) {
63         cout << "unbounded" << endl;
64     } else if (s.is_optimal()){
65         long result = ceil(CGAL::to_double(s.objective_value()));
66         cout << result << endl;
67     } else {
68         cout << "no" << endl;
69     }
70 }
71
72 int main() {
73     int p; cin >> p;
74 }
```

```
76     while (p > 0) {  
77         int a, b;  
78         cin >> a >> b;  
79         if (p == 1) solve_1(a, b);  
80         else if (p == 2) solve_2(a, b);  
81         cin >> p;  
82     }  
83 }
```

# Diet

## Keywords— LP/QP

```
1 #include <iostream>
2 #include <cassert>
3 #include <cmath>
4 #include <CGAL/basic.h>
5 #include <CGAL/QP_models.h>
6 #include <CGAL/QP_functions.h>
7 #include <CGAL/Gmpzf.h>
8
9 // choose exact integral type
10 typedef CGAL::Gmpzf ET;
11
12 // program and solution types
13 typedef CGAL::Quadratic_program<long> Program;
14 typedef CGAL::Quadratic_program_solution<ET> Solution;
15
16 using namespace std;
17
18 void diet(int n, int m){
19     Program lp (CGAL::LARGER, true, 0, false, 0);
20
21     for (int i = 0; i < n; i++) {
22         long c_min, c_max;
23         cin >> c_min >> c_max;
24
25         lp.set_b(i*2, c_min);
26         lp.set_b(i*2+1, c_max);
27         lp.set_r(i*2+1, CGAL::SMALLER);
28     }
29
30     for (int j = 0; j < m; j++) {
31         long p; cin >> p;
32         lp.set_c(j, p);
33         for (int i = 0; i < n; i++) {
34             long C; cin >> C;
35             lp.set_a(j, i*2, C);
36             lp.set_a(j, i*2+1, C);
37         }
38     }
39
40     Solution s = CGAL::solve_linear_program(lp, ET());
41     assert (s.solves_linear_program(lp));
42
43     if (s.is_infeasible() || s.is_unbounded()) {
44         cout << "No such diet." << endl;
45     } else {
46         long result = floor(CGAL::to_double(s.objective_value()));
47         cout << result << endl;
48     }
49 }
50
51
52
53 int main() {
54     int n, m;
55     cin >> n >> m;
56     while (!(n==0 && m==0)) {
57         diet(n, m);
58         cin >> n >> m;
59     }
60 }
```

## Portfolios

### Keywords— LP/QP

```
1 #include <iostream>
2 #include <cassert>
3 #include <cmath>
4 #include <CGAL/basic.h>
5 #include <CGAL/QP_models.h>
6 #include <CGAL/QP_functions.h>
7 #include <CGAL/Gmpzf.h>
8
9 // choose exact integral type
10 typedef CGAL::Gmpzf ET;
11
12 // program and solution types
13 typedef CGAL::Quadratic_program<long> Program;
14 typedef CGAL::Quadratic_program_solution<ET> Solution;
15
16 using namespace std;
17
18 bool solve_case(vector<long>& c, vector<long>& r, vector<vector<long>> >& v, long C, long R, long V
19 , int n) {
20     Program qp (CGAL::LARGER, true, 0, false, 0);
21     for (int i = 0; i < n; i++) {
22         qp.set_a(i, 0, r[i]);
23         qp.set_a(i, 1, c[i]);
24     }
25     qp.set_r(1, CGAL::SMALLER);
26     qp.set_b(0, R);
27     qp.set_b(1, C);
28
29     for (int i = 0; i < n; i++) {
30         for (int j = 0; j < n; j++) {
31             qp.set_d(i, j, 2*v[i][j]);
32         }
33     }
34
35     Solution s = CGAL::solve_quadratic_program(qp, ET());
36     assert (s.solves_quadratic_program(qp));
37
38     if (s.is_infeasible() || s.is_unbounded()) {
39         return false;
40     } else if (CGAL::to_double(s.objective_value()) <= V) {
41         return true;
42     } else {
43         return false;
44     }
45 }
46
47
48 void portfolios(int n, int m){
49     vector<long> c(n), r(n);
50     vector<vector<long>> > v(n, vector<long>(n));
51
52     for (int i = 0; i < n; i++) {
53         cin >> c[i] >> r[i];
54     }
55     for (int i = 0; i < n; i++) {
56         for (int j = 0; j < n; j++) {
57             cin >> v[i][j];
58         }
59     }
60     for (int i = 0; i < m; i++) {
61         long C, V, R;
62         cin >> C >> R >> V;
63         bool result = solve_case(c, r, v, C, R, V, n);
64         if (result) {
65             cout << "Yes." << endl;
66         } else {
67             cout << "No." << endl;
68         }
69     }
70 }
71
72 }
73
74
75 int main() {
76     int n, m;
```



```
77     while (true) {  
78         cin >> n >> m;  
79         if (n == 0 && m == 0) { break; }  
80         portfolios(n, m);  
81     }  
82 }
```

## Inball

*Keywords*— LP/QP

```
1 #include <iostream>
2 #include <cassert>
3 #include <cmath>
4 #include <CGAL/basic.h>
5 #include <CGAL/QP_models.h>
6 #include <CGAL/QP_functions.h>
7 #include <CGAL/Gmpz.h>
8
9 // choose exact integral type
10 typedef CGAL::Gmpz ET;
11
12 // program and solution types
13 typedef CGAL::Quadratic_program<long> Program;
14 typedef CGAL::Quadratic_program_solution<ET> Solution;
15
16 using namespace std;
17
18
19 void inball(int n){
20     int d; cin >> d;
21
22     Program lp (CGAL::SMALLER, false, 0, false, 0);
23     lp.set_c(d, -1);
24     lp.set_l(d, true, 0);
25     for (int i = 0; i < n; i++) {
26         long norm = 0;
27         for (int j = 0; j < d; j++) {
28             long a; cin >> a;
29             lp.set_a(j, i, a);
30             norm += a * a;
31         }
32         lp.set_a(d, i, sqrt(norm));
33         long b; cin >> b;
34         lp.set_b(i, b);
35     }
36
37     Solution s = CGAL::solve_linear_program(lp, ET());
38     assert (s.solves_linear_program(lp));
39
40     if (s.is_infeasible()) {
41         cout << "none" << endl;
42     } else if (s.is_unbounded()) {
43         cout << "inf" << endl;
44     } else {
45         long result = -CGAL::to_double(s.objective_value());
46         cout << result << endl;
47     }
48 }
49
50
51 int main() {
52     int n;
53     while (true) {
54         cin >> n;
55         if (n == 0) { break; }
56         inball(n);
57     }
58 }
```

# Knights

*Keywords*— Maximum Flow

```
1
2 #include <iostream>
3 #include <map>
4 #include <algorithm>
5 // BGL includes
6 #include <boost/graph/adjacency_list.hpp>
7 #include <boost/graph/push_relabel_max_flow.hpp>
8 // Namespaces
9 using namespace std;
10 using namespace boost;
11
12
13 // BGL Graph definitions
14 // =====
15 // Graph Type with nested interior edge properties for Flow Algorithms
16 typedef adjacency_list_traits<vecS, vecS, directedS> Traits;
17 typedef adjacency_list<vecS, vecS, directedS, no_property,
18     property<edge_capacity_t, long,
19     property<edge_residual_capacity_t, long,
20     property<edge_reverse_t, Traits::edge_descriptor> > > > Graph;
21 // Interior Property Maps
22 typedef property_map<Graph, edge_capacity_t>::type EdgeCapacityMap;
23 typedef property_map<Graph, edge_residual_capacity_t>::type ResidualCapacityMap;
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;
27 typedef graph_traits<Graph>::edge_iterator EdgeIt;
28
29
30 // Custom Edge Adder Class, that holds the references
31 // to the graph, capacity map and reverse edge map
32 // =====
33 class EdgeAdder {
34     Graph &G;
35     EdgeCapacityMap &capacitymap;
36     ReverseEdgeMap &revedgemap;
37
38 public:
39     // to initialize the Object
40     EdgeAdder(Graph & G, EdgeCapacityMap &capacitymap, ReverseEdgeMap &revedgemap):
41         G(G), capacitymap(capacitymap), revedgemap(revedgemap){}
42
43     // to use the Function (add an edge)
44     Edge addEdge(int from, int to, long capacity) {
45         Edge e, reverseE;
46         bool success;
47         tie(e, success) = add_edge(from, to, G);
48         tie(reverseE, success) = add_edge(to, from, G);
49         capacitymap[e] = capacity;
50         capacitymap[reverseE] = 0;
51         revedgemap[e] = reverseE;
52         revedgemap[reverseE] = e;
53         return e;
54     }
55 };
56
57
58 void knights() {
59     int m, n, k, c;
60     cin >> m >> n >> k >> c;
61
62     Graph G(2*m*n+2);
63     EdgeCapacityMap capacitymap = get(edge_capacity, G);
64     ReverseEdgeMap revedgemap = get(edge_reverse, G);
65     EdgeAdder eaG(G, capacitymap, revedgemap);
66
67     // Create graph representing the cave
68     Vertex src = vertex(2*n*m, G);
69     Vertex sink = vertex(2*n*m+1, G);
70     for (int i = 0; i < n; i++) {
71         for (int j = 0; j < m; j++) {
72             int pos_out = i*m + j;
73             int pos_in = pos_out + n*m;
74             int pos_out_right = pos_out + 1;
75             int pos_in_right = pos_out_right + n*m;
76             int pos_out_down = pos_out + m;
77             int pos_in_down = pos_out_down + n*m;
```

```

78 // Conect inner node with outer node
79 eaG.addEdge(pos_in, pos_out, c);
80 if (j < m-1) {
81     // Connecting node with the one on the right
82     eaG.addEdge(pos_out, pos_in_right, 1);
83     eaG.addEdge(pos_out_right, pos_in, 1);
84 }
85 if (i < n-1) {
86     // Connecting node with the one on the bottom
87     eaG.addEdge(pos_out, pos_in_down, 1);
88     eaG.addEdge(pos_out_down, pos_in, 1);
89 }
90 if (i == 0 || i == n-1) {
91     // Connecting top and bottom nodes to sink
92     eaG.addEdge(pos_out, sink, 1);
93 }
94 if (j == 0 || j == m-1) {
95     // Connecting left and right nodes to sink
96     eaG.addEdge(pos_out, sink, 1);
97 }
98 }
99 }
100
101 // Add edge from the source to the intersection representing the starting point of the knight
102 for (int i = 0; i < k; i++) {
103     int x, y; cin >> x >> y;
104     int pos = y*m+x + n*m;
105     eaG.addEdge(src, pos, 1);
106 }
107
108 // Compute the maximum number of knights as maximum flow
109 long flow = push_relabel_max_flow(G, src, sink);
110 cout << flow << endl;
111 }
112
113 int main() {
114     int T; cin >> T;
115     for (int t=0; t < T; t++){
116         knights();
117     }
118 }
119 }

```

## 2.4 Proximity Structures in CGAL

### Graypes

*Keywords*— Delaunay Triangulation

```
1  #include <vector>
2
3  #include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
4  #include <CGAL/Delaunay_triangulation_2.h>
5
6  typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
7  typedef CGAL::Delaunay_triangulation_2<K> Triangulation;
8  typedef Triangulation::Finite_faces_iterator Face_iterator;
9  typedef Triangulation::Edge_iterator EdgeIt;
10
11 using namespace std;
12
13
14 double ceil_to_double(const K::FT& x) {
15     double a = ceil(CGAL::to_double(x));
16     while (a+1 >= x) a -= 1;
17     while (a < x) a += 1;
18     return a;
19 }
20
21 void graypes(int n) {
22     vector<K::Point_2> p(n);
23     Triangulation t;
24
25     for (int i = 0; i < n; i++) {
26         long x, y;
27         cin >> x >> y;
28         p[i] = K::Point_2(x, y);
29     }
30     t.insert(p.begin(), p.end());
31
32     K::FT min_dist = t.segment(t.finite_edges_begin()).squared_length();
33     for (EdgeIt e = t.finite_edges_begin(); e != t.finite_edges_end(); e++) {
34         min_dist = min(min_dist, t.segment(e).squared_length());
35     }
36
37     K::FT d = sqrt(min_dist * (100 * 100) / (2 * 2));
38
39     cout << ceil_to_double(d) << endl;
40 }
41
42
43 int main() {
44     int n;
45     cout << fixed << setprecision(0);
46     while (true) {
47         cin >> n;
48         if (n == 0) { break; }
49         graypes(n);
50     }
51 }
```

## Bistro

*Keywords*— Delaunay Triangulation

```
1  #include <vector>
2
3  #include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
4  #include <CGAL/Delaunay_triangulation_2.h>
5
6  typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
7  typedef CGAL::Delaunay_triangulation_2<K> Triangulation;
8  typedef Triangulation::Finite_faces_iterator Face_iterator;
9  typedef Triangulation::Edge_iterator EdgeIt;
10 typedef K::Point_2 Point;
11
12 using namespace std;
13
14
15 double ceil_to_double(const K::FT& x) {
16     double a = ceil(CGAL::to_double(x));
17     while (a+1 >= x) a -= 1;
18     while (a < x) a += 1;
19     return a;
20 }
21
22 void bistro(int n) {
23     int m;
24     vector<Point> r(n);
25     Triangulation t;
26
27     for (int i = 0; i < n; i++) {
28         long x, y;
29         cin >> x >> y;
30         r[i] = Point(x, y);
31     }
32     t.insert(r.begin(), r.end());
33
34     cin >> m;
35
36     for (int i = 0; i < m; i++) {
37         long x, y;
38         cin >> x >> y;
39         Point p = Point(x,y);
40         K::FT dist = CGAL::squared_distance(p, t.nearest_vertex(p)->point());
41         cout << ceil_to_double(dist) << endl;
42     }
43 }
44
45
46
47 int main() {
48     int n;
49     cout << fixed << setprecision(0);
50     while (true) {
51         cin >> n;
52         if (n == 0) { break; }
53         bistro(n);
54     }
55 }
```

```

1  #include <vector>
2  #include <map>
3  #include <stack>
4
5  #include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
6  #include <CGAL/Delaunay_triangulation_2.h>
7
8  typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
9  typedef CGAL::Delaunay_triangulation_2<K> Triangulation;
10 typedef Triangulation::Finite_faces_iterator FaceIt;
11 typedef Triangulation::Edge_iterator EdgeIt;
12 typedef Triangulation::Face_handle FaceH;
13 typedef Triangulation::Vertex_handle VertexH;
14 typedef K::Point_2 Point;
15 typedef K::Segment_2 Segment;
16
17 using namespace std;
18
19
20 double ceil_to_double(const K::FT& x) {
21     double a = ceil(CGAL::to_double(x));
22     while (a+1 >= x) a -= 1;
23     while (a < x) a += 1;
24     return a;
25 }
26
27 void h1n1(int n) {
28
29     vector<Point> infected(n);
30     Triangulation t;
31     int m;
32
33     for (int i = 0; i < n; i++) {
34         long x, y;
35         cin >> x >> y;
36         infected[i] = Point(x, y);
37     }
38     t.insert(infected.begin(), infected.end());
39
40     cin >> m;
41     for (int i = 0; i < m; i++) {
42         long x, y, d;
43         cin >> x >> y >> d;
44         Point q(x, y);
45
46         bool result = false;
47
48         if (CGAL::squared_distance(q, t.nearest_vertex(q)->point()) < d) {
49             cout << "n";
50             continue;
51         }
52
53         FaceH face_1 = t.locate(q);
54
55         map<FaceH, bool> face_seen;
56         stack<FaceH> face_stack;
57
58         face_stack.push(face_1);
59         while(!face_stack.empty()) {
60             FaceH face = face_stack.top();
61             face_stack.pop();
62
63             // If you are placed on an infinite face, as you are not closer from d to the nearest
64             // point, you can escape from the infected
65             if (t.is_infinite(face)) {
66                 result = true;
67                 break;
68             }
69
70             face_seen[face] = true;
71
72             for (int j = 0; j < 3; j++) {
73                 Segment s = t.segment(face, j);
74
75                 // The point can not escape from this face through this edge of the triangulation
76                 if (s.squared_length() < d*4) { continue; }
77

```

```

78         // If can scape through, take the neighbour face
79         FaceH neighbour = face->neighbor(j);
80         // If it has been seen continue
81         if (face_seen[neighbour]) { continue; }
82         // Put to the queue of faces to study if scapatory
83         face_stack.push(neighbour);
84     }
85 }
86
87     if (result) {
88         cout << "y";
89     } else {
90         cout << "n";
91     }
92 }
93 cout << endl;
94 }
95
96
97
98 int main() {
99     int n;
100     cout << fixed << setprecision(0);
101     while (true) {
102         cin >> n;
103         if (n == 0) { break; }
104         h1n1(n);
105     }
106 }

```



## Germes

**Keywords**— Delaunay Triangulation, Nearest Neighbor

```
1  #include <vector>
2  #include <map>
3  #include <stack>
4  #include <algorithm>
5
6  #include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
7  #include <CGAL/Delaunay_triangulation_2.h>
8  #include <CGAL/nearest_neighbor_delaunay_2.h>
9
10 typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
11 typedef CGAL::Delaunay_triangulation_2<K> Triangulation;
12 typedef Triangulation::Finite_faces_iterator FaceIt;
13 typedef Triangulation::Edge_iterator EdgeIt;
14 typedef Triangulation::Finite_vertices_iterator VertexIt;
15 typedef Triangulation::Face_handle FaceH;
16 typedef Triangulation::Vertex_handle VertexH;
17 typedef K::Point_2 Point;
18 typedef K::Segment_2 Segment;
19
20 using namespace std;
21
22
23 double ceil_to_double(const K::FT& x) {
24     double a = ceil(CGAL::to_double(x));
25     while (a+1 >= x) a -= 1;
26     while (a < x) a += 1;
27     return a;
28 }
29
30 void germs(int n) {
31     long l, b, r, t;
32     cin >> l >> b >> r >> t;
33
34     vector<Point> bacteria(n);
35     vector<K::FT> distances(n);
36     for (int i = 0; i < n; i++) {
37         long x, y;
38         cin >> x >> y;
39         bacteria[i] = Point(x, y);
40     }
41
42     Triangulation tri;
43     tri.insert(bacteria.begin(), bacteria.end());
44
45     int i = 0;
46     for (VertexIt vi = tri.finite_vertices_begin(); vi != tri.finite_vertices_end(); vi++) {
47         VertexH vh = vi;
48         Point p = vh->point();
49
50         // Take the min distance to the nearest neighbour
51         K::FT min_distance;
52         if (tri.number_of_vertices() > 1) {
53             VertexH vn = CGAL::nearest_neighbor(tri, vh);
54             min_distance = CGAL::squared_distance(vh->point(), vn->point()) / 4;
55         } else {
56             // In the case the germ is alone, initialize min distance with the distance to one
57             // corner (it will always be greater than the distance to any of the boundaries)
58             min_distance = CGAL::squared_distance(vh->point(), Point(l, b));
59         }
60
61         // check with every boundary if the distance is less than the minimum
62         min_distance = min(min_distance, (p.x() - l) * (p.x() - l));
63         min_distance = min(min_distance, (p.x() - r) * (p.x() - r));
64         min_distance = min(min_distance, (p.y() - t) * (p.y() - t));
65         min_distance = min(min_distance, (p.y() - b) * (p.y() - b));
66
67         distances[i++] = min_distance;
68     }
69
70     sort(distances.begin(), distances.end());
71     vector<double> dead_time(n);
72     for (int i = 0; i < n; i++) {
73         K::FT d = distances[i];
74         long ti = ceil_to_double(sqrt(sqrt(d) - .5));
75         dead_time[i] = ti;
76     }
77 }
```

```

78 // Print results
79 cout << dead_time[0] << " " << dead_time[n/2] << " " << dead_time[n-1] << endl;
80
81 }
82
83
84 int main() {
85     int n;
86     cout << fixed << setprecision(0);
87     while (true) {
88         cin >> n;
89         if (n == 0) { break; }
90         germs(n);
91     }
92 }

```

# Stamps

## Keywords— LP/QP

```
1 #include <vector>
2 #include <cmath>
3
4 #include <CGAL/Exact_predicates_exact_constructions_kernel.h>
5 #include <CGAL/basic.h>
6 #include <CGAL/QP_models.h>
7 #include <CGAL/QP_functions.h>
8 #include <CGAL/Gmpz.h>
9
10 // Choose exact integral type
11 typedef CGAL::Gmpzf ET;
12 // Program and solution types
13 typedef CGAL::Quadratic_program<ET> Program;
14 typedef CGAL::Quadratic_program_solution<ET> Solution;
15 // Geometric types
16 typedef CGAL::Exact_predicates_exact_constructions_kernel K;
17 typedef K::Point_2 P;
18 typedef K::Segment_2 S;
19
20 using namespace std;
21
22
23 void stamps() {
24     int l, s, w; cin >> l >> s >> w;
25
26     vector<P> lamps(l), stamps(s);
27     vector<long> M(s);
28     vector<S> walls(w);
29
30     // Read data
31     for (int i = 0; i < l; i++) {
32         long x, y; cin >> x >> y;
33         lamps[i] = P(x, y);
34     }
35     for (int i = 0; i < s; i++) {
36         long x, y;
37         cin >> x >> y; stamps[i] = P(x, y);
38         cin >> M[i];
39     }
40     for (int i = 0; i < w; i++) {
41         long x_a, y_a, x_b, y_b;
42         cin >> x_a >> y_a >> x_b >> y_b;
43         walls[i] = S(P(x_a, y_a), P(x_b, y_b));
44     }
45
46     // Define quadratic program
47     Program lp (CGAL::SMALLER, true, 1, true, 1<<12);
48     for (int i = 0; i < s; i++) { // stamp index
49         for (int j = 0; j < l; j++) { // lamp index
50             S l_s(lamps[j], stamps[i]); // lamp to stamp segment
51
52             bool blocked = false;
53             for (int k = 0; k < w; k++) {
54                 if (CGAL::do_intersect(walls[k], l_s)) {
55                     blocked = true;
56                     break;
57                 }
58             }
59
60             if (!blocked) {
61                 double d = CGAL::to_double(1 / l_s.squared_length());
62                 lp.set_a(j, i, d);
63                 lp.set_a(j, i+s, d);
64             }
65         }
66         lp.set_b(i, M[i]);
67         lp.set_r(i, CGAL::SMALLER);
68         lp.set_b(s+i, 1);
69         lp.set_r(s+i, CGAL::LARGER);
70     }
71
72     // Get solution
73     Solution sol = CGAL::solve_linear_program(lp, ET());
74     assert (sol.solves_linear_program(lp));
75
76     if (sol.is_infeasible()) {
77         cout << "no" << endl;
```

```
78     } else {
79         cout << "yes" << endl;
80     }
81 }
82
83
84
85 int main() {
86     int t; cin >> t;
87     for (int i = 0; i < t; i++) {
88         stamps();
89     }
90 }
```

## 2.5 Advanced Flows

### Real Estate Market

*Keywords*— Max Flow Min Cost

```
1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4  // BGL includes
5  #include <boost/graph/adjacency_list.hpp>
6  #include <boost/graph/shortest_path_nonnegative_weights.hpp>
7  #include <boost/graph/find_flow_cost.hpp>
8  // Namespaces
9  using namespace std;
10 using namespace boost;
11
12
13 // BGL Graph definitions
14 // =====
15 // Graph Type with nested interior edge properties for Cost Flow Algorithms
16 typedef adjacency_list_traits<vecS, vecS, directedS> Traits;
17 typedef adjacency_list<vecS, vecS, directedS, no_property,
18     property<edge_capacity_t, long,
19     property<edge_residual_capacity_t, long,
20     property<edge_reverse_t, Traits::edge_descriptor,
21     property<edge_weight_t, long>>>> Graph;
22 // Interior Property Maps
23 typedef property_map<Graph, edge_capacity_t>::type EdgeCapacityMap;
24 typedef property_map<Graph, edge_weight_t>::type EdgeWeightMap;
25 typedef property_map<Graph, edge_residual_capacity_t>::type ResidualCapacityMap;
26 typedef property_map<Graph, edge_reverse_t>::type ReverseEdgeMap;
27 typedef graph_traits<Graph>::vertex_descriptor Vertex;
28 typedef graph_traits<Graph>::edge_descriptor Edge;
29 typedef graph_traits<Graph>::out_edge_iterator OutEdgeIt; // Iterator
30
31
32 // Custom Edge Adder Class, that holds the references
33 // to the graph, capacity map, weight map and reverse edge map
34 // =====
35 class EdgeAdder {
36     Graph &G;
37     EdgeCapacityMap &capacitymap;
38     EdgeWeightMap &weightmap;
39     ReverseEdgeMap &revedgemap;
40
41 public:
42     EdgeAdder(Graph & G, EdgeCapacityMap &capacitymap, EdgeWeightMap &weightmap, ReverseEdgeMap &
43         revedgemap)
44         : G(G), capacitymap(capacitymap), weightmap(weightmap), revedgemap(revedgemap) {}
45
46     void addEdge(int u, int v, long c, long w) {
47         Edge e, reverseE;
48         tie(e, tuples::ignore) = add_edge(u, v, G);
49         tie(reverseE, tuples::ignore) = add_edge(v, u, G);
50         capacitymap[e] = c;
51         weightmap[e] = w;
52         capacitymap[reverseE] = 0;
53         weightmap[reverseE] = -w;
54         revedgemap[e] = reverseE;
55         revedgemap[reverseE] = e;
56     }
57 };
58
59
60 void real_estate() {
61     int N, M, S;
62     cin >> N >> M >> S;
63
64     const int MAX = 100;
65
66     Graph G(N+M+S+2);
67     EdgeCapacityMap capacitymap = get(edge_capacity, G);
68     EdgeWeightMap weightmap = get(edge_weight, G);
69     ReverseEdgeMap revedgemap = get(edge_reverse, G);
70     ResidualCapacityMap rescapacitymap = get(edge_residual_capacity, G);
71     EdgeAdder ea(G, capacitymap, weightmap, revedgemap);
72
73     Vertex src = vertex(N+M+S, G);
74 }
```

```

75     Vertex sink = vertex(N+M+S+1, G);
76
77     // Store the graph
78     for (int i = 0; i < N; i++) { eaG.addEdge(src, i, 1, 0); }
79     for (int i = 0; i < S; i++) {
80         long l; cin >> l;
81         eaG.addEdge(N+M+i, sink, l, 0);
82     }
83     for (int i = 0; i < M; i++) {
84         int s; cin >> s;
85         eaG.addEdge(N+i, N+M+s-1, 1, 0);
86     }
87     for (int i = 0; i < N; i++) {
88         for (int j = 0; j < M; j++) {
89             long b; cin >> b;
90             eaG.addEdge(i, N+j, 1, MAX-b);
91         }
92     }
93
94     successive_shortest_path_nonnegative_weights(G, src, sink);
95     int cost = find_flow_cost(G);
96     int flow = 0;
97     // Iterate over all edges leaving the source to sum up the flow values.
98     OutEdgeIt e, eend;
99     for(tie(e, eend) = out_edges(vertex(src,G), G); e != eend; ++e) {
100         flow += capacitymap[*e] - rescapacitymap[*e];
101     }
102     cost = flow*MAX - cost;
103
104     cout << flow << " " << cost << endl;
105
106 }
107
108 int main() {
109     ios_base::sync_with_stdio(false);
110     int T; cin >> T;
111     for (int t=0; t < T; t++){
112         real_estate();
113     }
114 }

```

## Satellites

**Keywords**— Maximum Flow, Bipartite Matchings, DFS

```
1
2 #include <iostream>
3 #include <vector>
4 #include <queue>
5 // BGL includes
6 #include <boost/graph/adjacency_list.hpp>
7 #include <boost/graph/push_relabel_max_flow.hpp>
8 // Namespaces
9 using namespace std;
10 using namespace boost;
11
12
13 // BGL Graph definitions
14 // =====
15 // Graph Type with nested interior edge properties for Flow Algorithms
16 typedef adjacency_list_traits<vecS, vecS, directedS> Traits;
17 typedef adjacency_list<vecS, vecS, directedS, no_property,
18     property<edge_capacity_t, long,
19     property<edge_residual_capacity_t, long,
20     property<edge_reverse_t, Traits::edge_descriptor> > > > Graph;
21 // Interior Property Maps
22 typedef property_map<Graph, edge_capacity_t>::type      EdgeCapacityMap;
23 typedef property_map<Graph, edge_residual_capacity_t>::type ResidualCapacityMap;
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;
27 typedef graph_traits<Graph>::out_edge_iterator         OutEdgeIt;
28
29
30 // Custom Edge Adder Class, that holds the references
31 // to the graph, capacity map and reverse edge map
32 // =====
33 class EdgeAdder {
34     Graph &G;
35     EdgeCapacityMap &capacitymap;
36     ReverseEdgeMap &revedgemap;
37
38 public:
39     // to initialize the Object
40     EdgeAdder(Graph & G, EdgeCapacityMap &capacitymap, ReverseEdgeMap &revedgemap):
41         G(G), capacitymap(capacitymap), revedgemap(revedgemap){}
42
43     // to use the Function (add an edge)
44     void addEdge(int from, int to, long capacity) {
45         Edge e, reverseE;
46         bool success;
47         tie(e, success) = add_edge(from, to, G);
48         tie(reverseE, success) = add_edge(to, from, G);
49         capacitymap[e] = capacity;
50         capacitymap[reverseE] = 0;
51         revedgemap[e] = reverseE;
52         revedgemap[reverseE] = e;
53     }
54 };
55
56
57 void satellites() {
58     int g, s, l;
59     cin >> g >> s >> l;
60
61     Graph G(g+s+2);
62     EdgeCapacityMap capacitymap = get(edge_capacity, G);
63     ReverseEdgeMap revedgemap = get(edge_reverse, G);
64     ResidualCapacityMap rescapacitymap = get(edge_residual_capacity, G);
65     EdgeAdder eaG(G, capacitymap, revedgemap);
66
67     Vertex src = vertex(g+s, G);
68     Vertex sink = vertex(g+s+1, G);
69
70     int from, to;
71     for (int i = 0; i < l; i++) {
72         cin >> from >> to;
73         eaG.addEdge(from, g + to, 1);
74     }
75     for (int i = 0; i < g; i++) {
76         eaG.addEdge(src, i, 1);
77     }
```

```

78     for (int i = g; i < g + s; i++) {
79         eaG.addEdge(i, sink, 1);
80     }
81
82     push_relabel_max_flow(G, src, sink);
83
84     // BFS to find vertex set S
85     vector<int> vis(g+s+2, false); // visited flags
86     std::queue<int> Q; // BFS queue (from std:: not boost::)
87     vis[src] = true; // Mark the source as visited
88     Q.push(src);
89     while (!Q.empty()) {
90         const int u = Q.front();
91         Q.pop();
92         OutEdgeIt ebegin, eend;
93         for (tie(ebegin, eend) = out_edges(u, G); ebegin != eend; ++ebegin) {
94             const int v = target(*ebegin, G);
95             // Only follow edges with spare capacity
96             if (rescapacitymap[*ebegin] == 0 || vis[v]) continue;
97             vis[v] = true;
98             Q.push(v);
99         }
100     }
101
102     int count_g = 0, count_s = 0;
103     vector<int> result;
104     for (int i = 0; i < g; i++) {
105         if (!vis[i]) {
106             count_g++;
107             result.push_back(i);
108         }
109     }
110     for (int i = g; i < g + s; i++) {
111         if (vis[i]) {
112             count_s++;
113             result.push_back(i - g);
114         }
115     }
116
117     cout << count_g << " " << count_s << endl;
118     for (int i = 0; i < result.size(); i++) {
119         cout << result[i] << " ";
120         if (i == result.size() - 1) cout << endl;
121     }
122 }
123
124
125 int main() {
126     ios_base::sync_with_stdio(false);
127     int T; cin >> T;
128     for (int t=0; t < T; t++){
129         satellites();
130     }
131 }

```



```

1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4  #include <climits>
5  #include <queue>
6  // BGL includes
7  #include <boost/graph/adjacency_list.hpp>
8  #include <boost/graph/cycle_canceling.hpp>
9  #include <boost/graph/push_relabel_max_flow.hpp>
10 #include <boost/graph/successive_shortest_path_nonnegative_weights.hpp>
11 #include <boost/graph/find_flow_cost.hpp>
12 // Namespaces
13 using namespace std;
14 using namespace boost;
15
16
17 // BGL Graph definitions
18 // =====
19 // Graph Type with nested interior edge properties for Flow Algorithms
20 typedef adjacency_list_traits<vecS, vecS, directedS> Traits;
21 typedef adjacency_list<vecS, vecS, directedS, no_property,
22     property<edge_capacity_t, long,
23     property<edge_residual_capacity_t, long,
24     property<edge_reverse_t, Traits::edge_descriptor> > > > Graph;
25 // Interior Property Maps
26 typedef property_map<Graph, edge_capacity_t>::type EdgeCapacityMap;
27 typedef property_map<Graph, edge_residual_capacity_t>::type ResidualCapacityMap;
28 typedef property_map<Graph, edge_reverse_t>::type ReverseEdgeMap;
29 typedef graph_traits<Graph>::vertex_descriptor Vertex;
30 typedef graph_traits<Graph>::edge_descriptor Edge;
31 typedef graph_traits<Graph>::out_edge_iterator OutEdgeIt;
32
33 // Custom Edge Adder Class, that holds the references
34 // to the graph, capacity map and reverse edge map
35 // =====
36 class EdgeAdder {
37     Graph &G;
38     EdgeCapacityMap &capacitymap;
39     ReverseEdgeMap &revedgemap;
40
41 public:
42     // to initialize the Object
43     EdgeAdder(Graph & G, EdgeCapacityMap &capacitymap, ReverseEdgeMap &revedgemap):
44         G(G), capacitymap(capacitymap), revedgemap(revedgemap){}
45
46     // to use the Function (add an edge)
47     void addEdge(int from, int to, long capacity) {
48         Edge e, reverseE;
49         bool success;
50         tie(e, success) = add_edge(from, to, G);
51         tie(reverseE, success) = add_edge(to, from, G);
52         capacitymap[e] = capacity;
53         capacitymap[reverseE] = 0;
54         revedgemap[e] = reverseE;
55         revedgemap[reverseE] = e;
56     }
57 };
58
59
60 void algocoön() {
61     int n, m; cin >> n >> m;
62
63     Graph G(n);
64     EdgeCapacityMap capacitymap = get(edge_capacity, G);
65     ReverseEdgeMap revedgemap = get(edge_reverse, G);
66     ResidualCapacityMap rescapacitymap = get(edge_residual_capacity, G);
67     EdgeAdder eaG(G, capacitymap, revedgemap);
68
69     for (int i = 0; i < m; i++) {
70         int from, to, cost;
71         cin >> from >> to >> cost;
72         eaG.addEdge(from, to, cost);
73     }
74
75     int best_src = -1, best_sink = -1, best_value = numeric_limits<int>::max();
76     int flow;

```

```

78     for (int i = 1; i < n; i++) {
79         flow = push_relabel_max_flow(G, 0, i);
80         if (flow < best_value) {
81             best_value = flow; best_src = 0; best_sink = i;
82         }
83         flow = push_relabel_max_flow(G, i, 0);
84         if (flow < best_value) {
85             best_value = flow; best_src = i; best_sink = 0;
86         }
87     }
88
89     flow = push_relabel_max_flow(G, best_src, best_sink);
90
91     // BFS to find vertex set S
92     vector<bool> vis(n, false); // visited flags
93     std::queue<int> Q; // BFS queue (from std:: not boost::)
94     vis[best_src] = true; // Mark the source as visited
95     Q.push(best_src);
96     while (!Q.empty()) {
97         const int u = Q.front();
98         Q.pop();
99         OutEdgeIt ebeg, eend;
100         for (tie(ebeg, eend) = out_edges(u, G); ebeg != eend; ++ebeg) {
101             const int v = target(*ebeg, G);
102             // Only follow edges with spare capacity
103             if (rescapacitymap[*ebeg] == 0 || vis[v]) continue;
104             vis[v] = true;
105             Q.push(v);
106         }
107     }
108
109     cout << flow << endl;
110     cout << count(vis.begin(), vis.end(), true);
111     for (int i = 0; i < n; ++i) {
112         if (vis[i]) cout << " " << i;
113     }
114     cout << endl;
115 }
116
117 int main() {
118     ios_base::sync_with_stdio(false);
119     int T; cin >> T;
120     for (int t=0; t < T; t++){
121         algocoon();
122     }
123 }

```

## Canteen

*Keywords*— Max Flow Min Cost

```
1
2 #include <iostream>
3 #include <vector>
4 #include <algorithm>
5 #include <string>
6 // BGL includes
7 #include <boost/graph/adjacency_list.hpp>
8 #include <boost/graph/successive_shortest_path_nonnegative_weights.hpp>
9 #include <boost/graph/find_flow_cost.hpp>
10 // Namespaces
11 using namespace std;
12 using namespace boost;
13
14
15 // BGL Graph definitions
16 // =====
17 // Graph Type with nested interior edge properties for Cost Flow Algorithms
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,
23     property<edge_weight_t, long> > > > Graph;
24 // Interior Property Maps
25 typedef property_map<Graph, edge_capacity_t>::type EdgeCapacityMap;
26 typedef property_map<Graph, edge_weight_t>::type EdgeWeightMap;
27 typedef property_map<Graph, edge_residual_capacity_t>::type ResidualCapacityMap;
28 typedef property_map<Graph, edge_reverse_t>::type ReverseEdgeMap;
29 typedef graph_traits<Graph>::vertex_descriptor Vertex;
30 typedef graph_traits<Graph>::edge_descriptor Edge;
31 typedef graph_traits<Graph>::out_edge_iterator OutEdgeIt; // Iterator
32
33
34 // Custom Edge Adder Class, that holds the references
35 // to the graph, capacity map, weight map and reverse edge map
36 // =====
37 class EdgeAdder {
38     Graph &G;
39     EdgeCapacityMap &capacitymap;
40     EdgeWeightMap &weightmap;
41     ReverseEdgeMap &revedgemap;
42
43 public:
44     EdgeAdder(Graph & G, EdgeCapacityMap &capacitymap, EdgeWeightMap &weightmap, ReverseEdgeMap &
45         revedgemap)
46         : G(G), capacitymap(capacitymap), weightmap(weightmap), revedgemap(revedgemap) {}
47
48     void addEdge(int u, int v, long c, long w) {
49         Edge e, reverseE;
50         tie(e, tuples::ignore) = add_edge(u, v, G);
51         tie(reverseE, tuples::ignore) = add_edge(v, u, G);
52         capacitymap[e] = c;
53         weightmap[e] = w;
54         capacitymap[reverseE] = 0;
55         weightmap[reverseE] = -w;
56         revedgemap[e] = reverseE;
57         revedgemap[reverseE] = e;
58     }
59 };
60
61
62 void canteen() {
63     int n; cin >> n;
64
65     const int MAX_PRICE = 20;
66
67     vector<int> a(n), c(n), s(n), p(n), v(n-1), e(n-1);
68     for (int i = 0; i < n; i++) { cin >> a[i] >> c[i]; }
69     for (int i = 0; i < n; i++) { cin >> s[i] >> p[i]; }
70     for (int i = 0; i < n-1; i++) { cin >> v[i] >> e[i]; }
71
72     // Create Graph and Maps
73     Graph G(n+2);
74     EdgeCapacityMap capacitymap = get(edge_capacity, G);
75     EdgeWeightMap weightmap = get(edge_weight, G);
76     ReverseEdgeMap revedgemap = get(edge_reverse, G);
```

```

77 ResidualCapacityMap rescapacitymap = get(edge_residual_capacity, G);
78 EdgeAdder eaG(G, capacitymap, weightmap, revedgemap);
79
80 Vertex src = n, sink = n+1;
81 for (int i = 0; i < n; i++) {
82     eaG.addEdge(src, i, a[i], c[i]);
83     eaG.addEdge(i, sink, s[i], MAX_PRICE-p[i]);
84     if (i < n-1) { eaG.addEdge(i, i+1, v[i], e[i]); }
85 }
86
87 // Compute flow and cost
88 successive_shortest_path_nonnegative_weights(G, src, sink);
89 int cost = find_flow_cost(G);
90 int flow = 0;
91
92 for (int i = 0; i < n; i++) {
93     Edge ee; bool success;
94     tie(ee, success) = edge(i, sink, G);
95     int f = capacitymap[ee] - rescapacitymap[ee];
96     flow += f;
97     cost -= f * MAX_PRICE;
98 }
99 cost = -cost;
100
101 int total_s = accumulate(s.begin(), s.end(), 0);
102 string text = (total_s == flow) ? "possible " : "impossible ";
103 cout << text << flow << " " << cost << endl;
104 }
105
106 int main() {
107     ios_base::sync_with_stdio(false);
108     int T; cin >> T;
109     for (int t=0; t < T; t++){
110         canteen();
111     }
112 }

```

# Casino Royale

*Keywords*— Max Flow Min Cost

```
1
2 #include <iostream>
3 #include <vector>
4 #include <algorithm>
5 #include <string>
6 // BGL includes
7 #include <boost/graph/adjacency_list.hpp>
8 #include <boost/graph/successive_shortest_path_nonnegative_weights.hpp>
9 #include <boost/graph/find_flow_cost.hpp>
10 // Namespaces
11 using namespace std;
12 using namespace boost;
13
14
15 // BGL Graph definitions
16 // =====
17 // Graph Type with nested interior edge properties for Cost Flow Algorithms
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,
23     property<edge_weight_t, long> > > > Graph;
24 // Interior Property Maps
25 typedef property_map<Graph, edge_capacity_t>::type EdgeCapacityMap;
26 typedef property_map<Graph, edge_weight_t >::type EdgeWeightMap;
27 typedef property_map<Graph, edge_residual_capacity_t>::type ResidualCapacityMap;
28 typedef property_map<Graph, edge_reverse_t>::type ReverseEdgeMap;
29 typedef graph_traits<Graph>::vertex_descriptor Vertex;
30 typedef graph_traits<Graph>::edge_descriptor Edge;
31 typedef graph_traits<Graph>::out_edge_iterator OutEdgeIt; // Iterator
32
33
34 // Custom Edge Adder Class, that holds the references
35 // to the graph, capacity map, weight map and reverse edge map
36 // =====
37 class EdgeAdder {
38     Graph &G;
39     EdgeCapacityMap &capacitymap;
40     EdgeWeightMap &weightmap;
41     ReverseEdgeMap &revedgemap;
42
43 public:
44     EdgeAdder(Graph & G, EdgeCapacityMap &capacitymap, EdgeWeightMap &weightmap, ReverseEdgeMap &
45         revedgemap)
46         : G(G), capacitymap(capacitymap), weightmap(weightmap), revedgemap(revedgemap) {}
47
48     Edge addEdge(int u, int v, long c, long w) {
49         Edge e, reverseE;
50         tie(e, tuples::ignore) = add_edge(u, v, G);
51         tie(reverseE, tuples::ignore) = add_edge(v, u, G);
52         capacitymap[e] = c;
53         weightmap[e] = w;
54         capacitymap[reverseE] = 0;
55         weightmap[reverseE] = -w;
56         revedgemap[e] = reverseE;
57         revedgemap[reverseE] = e;
58         return e;
59     }
60 };
61
62 void casino_royale() {
63     int n, m, l;
64     cin >> n >> m >> l;
65
66     const int MAX_PRIORITY = 1<<7;
67
68     // Create Graph and Maps
69     Graph G(n+2);
70     EdgeCapacityMap capacitymap = get(edge_capacity, G);
71     EdgeWeightMap weightmap = get(edge_weight, G);
72     ReverseEdgeMap revedgemap = get(edge_reverse, G);
73     ResidualCapacityMap rescapacitymap = get(edge_residual_capacity, G);
74     EdgeAdder eaG(G, capacitymap, weightmap, revedgemap);
75     Vertex v_source = n, v_target = n + 1;
76
77     for (int i = 0; i < n-1; i++) { eaG.addEdge(i, i+1, l, MAX_PRIORITY); }
```

```

77     eaG.addEdge(v_source, 0, 1, 0);
78     eaG.addEdge(n-1, v_target, 1, 0);
79
80     // Add the missions edges
81     int x, y, q;
82     for (int i = 0; i < m; i++) {
83         cin >> x >> y >> q;
84         eaG.addEdge(x, y, 1, (MAX_PRIORITY*(y-x) - q));
85     }
86
87     // Find MaxFlowMinCost
88     successive_shortest_path_nonnegative_weights(G, v_source, v_target);
89     int cost = find_flow_cost(G), flow = 0;
90     OutEdgeIt e, eend;
91     for(tie(e, eend) = out_edges(vertex(v_source,G), G); e != eend; ++e) {
92         flow += capacitymap[*e] - rescapacitymap[*e];
93     }
94     cost -= MAX_PRIORITY * (n-1) * flow;
95     cout << -cost << endl;
96 }
97
98 int main() {
99     ios_base::sync_with_stdio(false);
100     int T; cin >> T;
101     for (int t=0; t < T; t++){
102         casino_royale();
103     }
104 }

```

# Chapter 3

## Exam Preparation

### Odd Route

*Keywords*— Dijkstra Shortest Path

```
1 // STL includes
2 #include <vector>
3 #include <iostream>
4 // BGL includes
5 #include <boost/graph/adjacency_list.hpp>
6 #include <boost/graph/dijkstra_shortest_paths.hpp>
7
8 // Namespaces
9 using namespace std;
10 using namespace boost;
11
12 // Directed graph with integer weights on edges.
13 typedef adjacency_list<vecS, vecS, directedS, 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 // Property map edge -> weight
17 typedef property_map<Graph, edge_weight_t>::type WeightMap;
18
19
20 void odd_route() {
21     int n, m; cin >> n >> m;
22     int s, t; cin >> s >> t;
23
24     Graph G(4*n);
25     WeightMap weightmap = get(edge_weight, G);
26
27     /*
28     Split all vertices u, 0 <= u < V(G) into 4 parts
29     4*u (even length , even weight)
30     4*u+1 (even length, odd weight)
31     4*u+2 (odd length, even weight)
32     4*u+3 (odd length, odd weight)
33     */
34
35     Edge e;
36     int u, v, w;
37     for (int i = 0; i < m; i++) {
38         cin >> u >> v >> w;
39         tie(e, tuples::ignore) = add_edge(4*u, 4*v + 2 + w%2, G); weightmap[e] = w;
40         tie(e, tuples::ignore) = add_edge(4*u + 1, 4*v + 3 - w%2, G); weightmap[e] = w;
41         tie(e, tuples::ignore) = add_edge(4*u + 2, 4*v + w%2, G); weightmap[e] = w;
42         tie(e, tuples::ignore) = add_edge(4*u + 3, 4*v + 1 - w%2, G); weightmap[e] = w;
43     }
44
45     vector<int> dist(4 * n), pred(4 * n);
46     dijkstra_shortest_paths(G, 4*s,
47                             predecessor_map(make_iterator_property_map(pred.begin(),
48                                     get(vertex_index, G)))
49                             .distance_map(make_iterator_property_map(dist.begin(),
50                                     get(vertex_index, G)))
51                             )
52     );
53
54     if (dist[4*t + 3] == INT_MAX)
55         cout << "no" << endl;
56     else
57         cout << dist[4*t + 3] << endl;
58 }
59 }
```

```
60
61 int main() {
62     ios_base::sync_with_stdio(false);
63     int T; cin >> T;
64     for (int t=0; t < T; t++){
65         odd_route();
66     }
67 }
```



## Light the Stage

**Keywords**— Delaunay Triangulation, Nearest Neighbor

```
1 #include <vector>
2 #include <iostream>
3 #include <algorithm>
4 #include <limits>
5
6 #include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
7 #include <CGAL/Delaunay_triangulation_2.h>
8
9 typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
10 typedef CGAL::Delaunay_triangulation_2<K> Triangulation;
11 typedef K::Point_2 P;
12
13 using namespace std;
14
15
16 void light_the_stage() {
17     int m, n, h;
18     cin >> m >> n;
19
20     vector<P> l(n), p(m);
21     vector<int> r(m);
22     vector<int> dead_time(m, INT_MAX);
23
24     // Read input
25     int x, y;
26     for (int i = 0; i < m; i++) {
27         cin >> x >> y;
28         p[i] = P(x, y);
29         cin >> r[i];
30     }
31     cin >> h;
32     for (int i = 0; i < n; i++) {
33         cin >> x >> y;
34         l[i] = P(x, y);
35     }
36
37     // Create triangulation for the lamps points
38     Triangulation tri;
39     tri.insert(l.begin(), l.end());
40
41     // Iterate over all participants and look for the closest lamp
42     for (int i = 0; i < m; i++) {
43         P v = tri.nearest_vertex(p[i])->point();
44         double d_2 = CGAL::squared_distance(v, p[i]);
45         double d = (r[i] + h);
46         double d_max = d * d;
47
48         if (d_max <= d_2) continue;
49
50         for (int j = 0; j < n; j++) {
51             double dist = CGAL::squared_distance(p[i], l[j]);
52             if (d_max > dist) {
53                 dead_time[i] = j;
54                 break;
55             }
56         }
57     }
58
59     int winner_time = *std::max_element(dead_time.begin(), dead_time.end());
60
61     // Print result
62     for (int i = 0; i < m; i++) {
63         if (dead_time[i] == winner_time) cout << i << " ";
64     }
65     cout << endl;
66 }
67
68
69 int main() {
70     std::ios_base::sync_with_stdio(false);
71     int t; cin >> t;
72     for (int i = 0; i < t; i++) {
73         light_the_stage();
74     }
75 }
```

## Bonus Level

*Keywords*— Max Flow Min Cost

```
1 // Includes
2 // =====
3 // STL includes
4 #include <iostream>
5 #include <cstdlib>
6 // BGL includes
7 #include <boost/graph/adjacency_list.hpp>
8 #include <boost/graph/successive_shortest_path_nonnegative_weights.hpp>
9 #include <boost/graph/find_flow_cost.hpp>
10 // Namespaces
11 using namespace boost;
12 using namespace std;
13
14 // BGL Graph definitions
15 // =====
16 // Graph Type with nested interior edge properties for Cost Flow Algorithms
17 typedef adjacency_list_traits<vecS, vecS, directedS> Traits;
18 typedef adjacency_list<vecS, vecS, directedS, no_property,
19     property<edge_capacity_t, long,
20     property<edge_residual_capacity_t, long,
21     property<edge_reverse_t, Traits::edge_descriptor,
22     property<edge_weight_t, long> > > > > Graph;
23 // Interior Property Maps
24 typedef property_map<Graph, edge_capacity_t>::type EdgeCapacityMap;
25 typedef property_map<Graph, edge_weight_t>::type EdgeWeightMap;
26 typedef property_map<Graph, edge_residual_capacity_t>::type ResidualCapacityMap;
27 typedef property_map<Graph, edge_reverse_t>::type ReverseEdgeMap;
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; // Iterator
31
32 // Custom Edge Adder Class, that holds the references
33 // to the graph, capacity map, weight map and reverse edge map
34 // =====
35 class EdgeAdder {
36     Graph &G;
37     EdgeCapacityMap &capacitymap;
38     EdgeWeightMap &weightmap;
39     ReverseEdgeMap &revedgemap;
40
41 public:
42     EdgeAdder(Graph & G, EdgeCapacityMap &capacitymap, EdgeWeightMap &weightmap, ReverseEdgeMap &
43         revedgemap)
44         : G(G), capacitymap(capacitymap), weightmap(weightmap), revedgemap(revedgemap) {}
45
46     void addEdge(int u, int v, long c, long w) {
47         Edge e, reverseE;
48         tie(e, tuples::ignore) = add_edge(u, v, G);
49         tie(reverseE, tuples::ignore) = add_edge(v, u, G);
50         capacitymap[e] = c;
51         weightmap[e] = w;
52         capacitymap[reverseE] = 0;
53         weightmap[reverseE] = -w;
54         revedgemap[e] = reverseE;
55         revedgemap[reverseE] = e;
56     }
57 };
58
59 void bonus_level() {
60     int n; cin >> n;
61
62     const int MAX_COST = 100;
63     // Create Graph and Maps
64     Graph G(3 * n * n);
65     EdgeCapacityMap capacitymap = get(edge_capacity, G);
66     EdgeWeightMap weightmap = get(edge_weight, G);
67     ReverseEdgeMap revedgemap = get(edge_reverse, G);
68     EdgeAdder eaG(G, capacitymap, weightmap, revedgemap);
69
70     Vertex src = add_vertex(G);
71     Vertex sink = add_vertex(G);
72
73     int a, pos;
74     for (int i = 0; i < n; i++) {
75         for (int j = 0; j < n; j++) {
76             cin >> a;
77             pos = 3 * j + 3 * n * i;
```

```

77     eaG.addEdge(pos, pos + 1, 1, MAX_COST - a);
78     eaG.addEdge(pos, pos + 2, 1, MAX_COST);
79     if (i != n - 1) {
80         eaG.addEdge(pos + 1, pos + 3 * n, 1, 0);
81         eaG.addEdge(pos + 2, pos + 3 * n, 1, 0);
82     }
83     if (j != n - 1) {
84         eaG.addEdge(pos + 1, pos + 3, 1, 0);
85         eaG.addEdge(pos + 2, pos + 3, 1, 0);
86     }
87 }
88 }
89 eaG.addEdge(src, 0, 2, 0);
90 eaG.addEdge(3 * n * n - 1, sink, 1, 0);
91 eaG.addEdge(3 * n * n - 2, sink, 1, 0);
92
93 successive_shortest_path_nonnegative_weights(G, src, sink);
94 int cost = find_flow_cost(G);
95 cout << - cost + 2 * (2 * n - 1) * MAX_COST << endl;
96
97 }
98
99 int main() {
100     int T; cin >> T;
101     for (int t = 0; t < T; t++){
102         bonus_level();
103     }
104 }

```

```

1 // Includes
2 // =====
3 // STL includes
4 #include <iostream>
5 #include <vector>
6 #include <algorithm>
7 #include <climits>
8 // BGL includes
9 #include <boost/graph/adjacency_list.hpp>
10 #include <boost/graph/connected_components.hpp>
11 // CGAL
12 #include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
13 #include <CGAL/Delaunay_triangulation_2.h>
14 #include <CGAL/Triangulation_vertex_base_with_info_2.h>
15
16 // Namespaces
17 using namespace std;
18 using namespace boost;
19
20 // Directed graph with integer weights on edges.
21 typedef adjacency_list<vecS, vecS, undirectedS,
22     no_property, property<edge_weight_t, int>> Graph;
23 typedef graph_traits<Graph>::vertex_descriptor Vertex; // Vertex type
24 typedef graph_traits<Graph>::edge_descriptor Edge; // Edge type
25 typedef graph_traits<Graph>::edge_iterator EdgeIt; // Edge iterator
26 // Property map edge -> weight
27 typedef property_map<Graph, edge_weight_t>::type WeightMap;
28
29 typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
30 typedef K::Point_2 P;
31
32 typedef CGAL::Triangulation_vertex_base_with_info_2<int,K> Vb;
33 typedef CGAL::Triangulation_face_base_2<K> Fb;
34 typedef CGAL::Triangulation_data_structure_2<Vb,Fb> Tds;
35 typedef CGAL::Delaunay_triangulation_2<K,Tds> Triangulation;
36 typedef Triangulation::Edge_iterator Edge_iterator;
37 typedef Triangulation::Vertex_handle VertexH;
38
39 int get_k_planets(int t, vector<pair<P, int>> & planets, const int r, const int n) {
40     assert(t > 0);
41     assert(t < n);
42     int n_not_conquered = n - t;
43
44     Graph G(n_not_conquered);
45
46     Triangulation tr;
47     tr.insert(planets.begin(), planets.begin() + n_not_conquered);
48
49     for (Edge_iterator e = tr.finite_edges_begin(); e != tr.finite_edges_end(); e++) {
50         VertexH v1 = e->first->vertex((e->second + 1) % 3);
51         VertexH v2 = e->first->vertex((e->second + 2) % 3);
52         double dist = CGAL::squared_distance(v1->point(), v2->point());
53         double rr = r;
54         double min_dist = rr * rr;
55         if (dist <= min_dist) {
56             add_edge(v1->info(), v2->info(), G);
57         }
58     }
59
60     // Analyze components
61     vector<int> component(n_not_conquered);
62     int n_components = connected_components(G, &component[0]);
63     vector<int> sizes(n_components, 0);
64     int largest = -1;
65     for (int i = 0; i < n_not_conquered; i++) {
66         int c = component[i];
67         sizes[c]++;
68         largest = max(largest, sizes[c]);
69     }
70
71     return min(t, largest);
72 }
73
74 int search_t(int begin, int end, vector<pair<P, int>> & planets, const int r, const int n) {
75     if (begin + 1 == end)
76         return max(get_k_planets(begin, planets, r, n), get_k_planets(end, planets, r, n));
77     else {

```

```

78     int middle = (begin + end) / 2;
79     int b = get_k_planets(middle, planets, r, n);
80     if (b == middle) return search_t(middle, end, planets, r, n);
81     else return search_t(begin, middle, planets, r, n);
82 }
83 }
84
85 void sith() {
86     int n, r; cin >> n >> r;
87
88     vector<pair<P, int> > p(n);
89     long x, y;
90     for (int i = n-1; i >= 0; i--) {
91         cin >> x >> y;
92         p[i] = make_pair(P(x, y), i);
93     }
94
95     int t = search_t(0, n/2, p, r, n);
96     cout << t << endl;
97 }
98
99
100 int main() {
101     int T; cin >> T;
102     for (int t = 0; t < T; t++){
103         sith();
104     }
105 }

```

## Clues

**Keywords**— Delaunay Triangulation, Nearest Neighbor

```
1 #include <vector>
2 #include <map>
3 #include <stack>
4
5 #include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
6 #include <CGAL/Delaunay_triangulation_2.h>
7 #include <CGAL/Triangulation_vertex_base_with_info_2.h>
8
9 using namespace std;
10
11 // CGAL typedefs
12 typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
13 typedef pair<int, bool> info_t;
14 typedef CGAL::Triangulation_vertex_base_with_info_2<info_t, K> Vb;
15 typedef CGAL::Triangulation_data_structure_2<Vb> Tds;
16 typedef CGAL::Delaunay_triangulation_2<K, Tds> Triangulation;
17 typedef Triangulation::Finite_faces_iterator FaceIt;
18 typedef Triangulation::Edge_iterator EdgeIt;
19 typedef Triangulation::Face_handle FaceH;
20 typedef Triangulation::Vertex_handle VertexH;
21 typedef Triangulation::Vertex_circulator VertexCirculator;
22 typedef K::Point_2 Point;
23 typedef K::Segment_2 Segment;
24 typedef K::Triangle_2 Triangle;
25
26
27 bool has_interference(Triangulation const & trg, K::FT const & rr) {
28     for (auto e = trg.finite_edges_begin(); e != trg.finite_edges_end(); e++)
29         if (trg.segment(*e).squared_length() <= rr) return true;
30     return false;
31 }
32
33 bool try_two_colors(Triangulation & trg, K::FT const & rr) {
34     for (auto v = trg.finite_vertices_begin(); v != trg.finite_vertices_end(); v++)
35         v->info() = { 0, false };
36
37     int components = 0;
38     Triangulation trg0, trg1;
39     for (auto v = trg.finite_vertices_begin(); v != trg.finite_vertices_end(); v++) {
40         if (v->info().first == 0) {
41             v->info().first = ++components;
42             vector<VertexH> stack(1, v);
43             do {
44                 VertexH h = stack.back();
45                 stack.pop_back();
46                 VertexCirculator c = trg.incident_vertices(h);
47                 do if (!trg.is_infinite(c) && CGAL::squared_distance(h->point(), c->point()) <= rr) {
48                     if (c->info() == h->info()) return false;
49                     if (c->info().first == 0) {
50                         stack.push_back(c);
51                         c->info() = { components, !h->info().second };
52                     }
53                 } while (++c != trg.incident_vertices(h));
54             } while (!stack.empty());
55         }
56
57         if (v->info().second) trg1.insert(v->point());
58         else trg0.insert(v->point());
59     }
60
61     return !has_interference(trg0, rr) && !has_interference(trg1, rr);
62 }
63
64 void clues() {
65     int n, m;
66     long r;
67     cin >> n >> m >> r;
68     K::FT rr(r*r);
69
70     vector<Point> stations(n);
71     for (int i = 0; i < n; i++) cin >> stations[i];
72
73     Triangulation trg;
74     trg.insert(stations.begin(), stations.end());
75     bool success = try_two_colors(trg, rr);
76 }
```

```

77     for (int i = 0; i < m; i++) {
78         Point holmes, watson;
79         cin >> holmes >> watson;
80
81         if (success) {
82             if (CGAL::squared_distance(holmes, watson) <= rr) {
83                 cout << "y"; continue;
84             }
85
86             auto holmes_station = trg.nearest_vertex(holmes);
87             auto watson_station = trg.nearest_vertex(watson);
88             if (holmes_station->info().first == watson_station->info().first &&
89                 CGAL::squared_distance(holmes_station->point(), holmes) <= rr &&
90                 CGAL::squared_distance(watson_station->point(), watson) <= rr)
91             {
92                 cout << "y"; continue;
93             }
94         }
95         cout << "n";
96     }
97     cout << endl;
98 }
99
100
101
102 int main() {
103     ios_base::sync_with_stdio(false);
104     int T; cin >> T;
105     for (int t = 0; t < T; t++) {
106         clues();
107     }
108 }

```

# Punch

## Keywords—

```
1 #include <iostream>
2 #include <cassert>
3 #include <CGAL/basic.h>
4 #include <CGAL/QP_models.h>
5 #include <CGAL/QP_functions.h>
6 #include <CGAL/Gmpz.h>
7
8 // choose exact integral type
9 typedef CGAL::Gmpz ET;
10
11 // program and solution types
12 typedef CGAL::Quadratic_program<int> Program;
13 typedef CGAL::Quadratic_program_solution<ET> Solution;
14
15 using namespace std;
16
17 void punch(){
18     int n, k; cin >> n >> k;
19
20     Program qp (CGAL::EQUAL, true, 0, false, 0);
21     int c, v;
22     for (int i = 0; i < n; i++) {
23         cin >> c >> v;
24         qp.set_c(i, c);
25         qp.set_a(i, 0, v);
26         // qp.set_d(i, i, 1);
27     }
28     qp.set_b(0, k);
29
30     Solution s = CGAL::solve_linear_program(qp, ET());
31     assert(s.solves_linear_program(qp));
32     assert(s.is_optimal());
33
34     // CGAL::Quadratic_program_solution<ET>::Variable_value_iterator opt = s.variable_values_begin
35     // ();
36     // int value = 0;
37     // for (; opt != s.variable_values_end(); opt++) {
38     //     int tmp = int(CGAL::to_double(*(opt)));
39     //     if (tmp > 0) value++;
40     // }
41     // cout << value << endl;
42     cout << CGAL::to_double(s.objective_value()) << endl;
43 }
44
45
46 int main() {
47     ios_base::sync_with_stdio(false);
48     int T; cin >> T;
49
50     for (int i = 0; i < T; i++) {
51         punch();
52     }
53 }
```



## Carsharing

*Keywords*— Max Flow Min Cost

```
1 #include <iostream>
2 #include <vector>
3 #include <set>
4 #include <map>
5 // BGL include
6 #include <boost/graph/adjacency_list.hpp>
7 #include <boost/graph/successive_shortest_path_nonnegative_weights.hpp>
8 #include <boost/graph/find_flow_cost.hpp>
9
10 using namespace std;
11 using namespace boost;
12
13 // BGL Graph definitions
14 // =====
15 // Graph Type with nested interior edge properties for Cost Flow Algorithms
16 typedef adjacency_list_traits<vecS, vecS, directedS> Traits;
17 typedef adjacency_list<vecS, vecS, directedS, no_property,
18     property<edge_capacity_t, long,
19     property<edge_residual_capacity_t, long,
20     property<edge_reverse_t, Traits::edge_descriptor,
21     property<edge_weight_t, long> > > > > Graph;
22 // Interior Property Maps
23 typedef property_map<Graph, edge_capacity_t>::type EdgeCapacityMap;
24 typedef property_map<Graph, edge_weight_t>::type EdgeWeightMap;
25 typedef property_map<Graph, edge_residual_capacity_t>::type ResidualCapacityMap;
26 typedef property_map<Graph, edge_reverse_t>::type ReverseEdgeMap;
27 typedef graph_traits<Graph>::vertex_descriptor Vertex;
28 typedef graph_traits<Graph>::edge_descriptor Edge;
29 typedef graph_traits<Graph>::out_edge_iterator OutEdgeIt; // Iterator
30
31 // Custom Edge Adder Class, that holds the references
32 // to the graph, capacity map, weight map and reverse edge map
33 // =====
34 class EdgeAdder {
35     Graph &G;
36     EdgeCapacityMap &capacitymap;
37     EdgeWeightMap &weightmap;
38     ReverseEdgeMap &revedgemap;
39
40 public:
41     EdgeAdder(Graph & G, EdgeCapacityMap &capacitymap, EdgeWeightMap &weightmap, ReverseEdgeMap &
42         revedgemap)
43         : G(G), capacitymap(capacitymap), weightmap(weightmap), revedgemap(revedgemap) {}
44
45     void addEdge(int u, int v, long c, long w) {
46         Edge e, reverseE;
47         tie(e, tuples::ignore) = add_edge(u, v, G);
48         tie(reverseE, tuples::ignore) = add_edge(v, u, G);
49         capacitymap[e] = c;
50         weightmap[e] = w;
51         capacitymap[reverseE] = 0;
52         weightmap[reverseE] = -w;
53         revedgemap[e] = reverseE;
54         revedgemap[reverseE] = e;
55     }
56 };
57
58 struct Booking {
59     int s, t, d, a, p;
60 };
61
62 void carsharing() {
63     int N, S;
64     cin >> N >> S;
65
66     const int MAXL = 100, MAXT = 100000, MAXP = 100;
67     const int INF = MAXL*S;
68
69     // Define variables to store data
70     vector<int> L(S);
71     vector<Booking> B;
72     vector<set<int> > times(S);
73     vector<map<int, int> > M(S);
74     vector<int> psum(S+1);
75
76     // Read data
77     for (int i = 0; i < S; i++) {
```

```

77     cin >> L[i];
78     times[i].insert(0); times[i].insert(MAXT);
79 }
80
81 int s, t, d, a, p;
82 for (int i = 0; i < N; i++) {
83     cin >> s >> t >> d >> a >> p;
84     s--; t--;
85     times[s].insert(d); times[t].insert(a);
86     B.push_back({s,t,d,a,p});
87 }
88
89 for (int s = 0; s < S; s++) {
90     int i = 0;
91     for (auto &t : times[s]) {
92         M[s][t] = i;
93         i++;
94     }
95     psum[s+1] = psum[s] + M[s].size();
96 }
97
98 int T = psum.back();
99 int v_source = T, v_target = T + 1;
100 Graph G(T+2);
101 EdgeCapacityMap capacitymap = get(edge_capacity, G);
102 EdgeWeightMap weightmap = get(edge_weight, G);
103 ReverseEdgeMap revedgemap = get(edge_reverse, G);
104 ResidualCapacityMap rescapacitymap = get(edge_residual_capacity, G);
105 EdgeAdder eaG(G, capacitymap, weightmap, revedgemap);
106
107 for (int i = 0; i < S; i++) {
108     eaG.addEdge(v_source, psum[i], L[i], 0);
109     eaG.addEdge(psum[i+1]-1, v_target, INF, 0);
110     int it = -1, lastt = 0;
111     for (auto &t : times[i]) {
112         if (it != -1) {
113             eaG.addEdge(psum[i]+it, psum[i]+it+1, INF, MAXP*(t-lastt));
114         }
115         it++; lastt = t;
116     }
117 }
118
119 for (int i = 0; i < N; i++) {
120     eaG.addEdge(psum[B[i].s] + M[B[i].s][B[i].d],
121                 psum[B[i].t] + M[B[i].t][B[i].a],
122                 1, ((B[i].a - B[i].d) * MAXP) - B[i].p);
123 }
124
125 successive_shortest_path_nonnegative_weights(G, v_source, v_target);
126 int flow = 0;
127 // Iterate over all edges leaving the source to sum up the flow values.
128 OutEdgeIt e, eend;
129 for (tie(e, eend) = out_edges(vertex(v_source, G), G); e != eend; ++e) {
130     flow += capacitymap[*e] - rescapacitymap[*e];
131 }
132 int cost = MAXP * MAXT * flow - find_flow_cost(G);
133 cout << cost << endl;
134 }
135
136 int main() {
137     ios_base::sync_with_stdio(false);
138     int T; cin >> T;
139     for (int t = 0; t < T; t++) {
140         carsharing();
141     }
142 }

```

## Planks

**Keywords**— Backtracking, Split and List

```
1 #include <algorithm>
2 #include <vector>
3 #include <iostream>
4
5 using namespace std;
6
7 typedef vector<vector<int>> > vecvec;
8
9 void back_track(vector<int>& planks, vecvec& assignment, vecvec& F, int id, int ubound) {
10     if (id >= ubound) {
11         vector<int> tuple(4, 0);
12         for (int i = 0; i < 4; i++) {
13             for (int j = 0; j < assignment[i].size(); j++)
14                 tuple[i] += planks[assignment[i][j]];
15         }
16         F.push_back(tuple);
17         return;
18     }
19
20     for (int side = 0; side < 4; side++) {
21         assignment[side].push_back(id);
22         back_track(planks, assignment, F, id+1, ubound);
23         assignment[side].pop_back();
24     }
25 }
26
27 void planks() {
28     int n; cin >> n;
29
30     vector<int> planks(n);
31     int sum = 0;
32     long long n_sol = 0;
33
34     for (int i = 0; i < n; i++) { cin >> planks[i]; sum += planks[i]; }
35
36     if (sum % 4 != 0) {
37         cout << 0 << endl;
38         return;
39     }
40
41     vecvec F1, F2, assignment1(4), assignment2(4);
42     back_track(planks, assignment1, F1, 0, n/2);
43     back_track(planks, assignment2, F2, n/2, n);
44     sort(F2.begin(), F2.end());
45
46     for (int idx = 0; idx < F1.size(); idx++) {
47         vector<int> member = F1[idx];
48         for (int i = 0; i < 4; i++)
49             member[i] = sum/4 - member[i];
50
51         pair<vecvec::iterator, vecvec::iterator> bounds;
52         bounds = equal_range(F2.begin(), F2.end(), member);
53         n_sol += distance(bounds.first, bounds.second);
54     }
55
56     cout << n_sol / 24 << endl;
57 }
58
59 int main() {
60     ios_base::sync_with_stdio(false);
61     int T; cin >> T;
62     for (int t=0; t < T; t++){
63         planks();
64     }
65 }
```

## New Tiles

*Keywords*—

```
1
2  #include <iostream>
3  #include <vector>
4  #include <algorithm>
5
6  using namespace std;
7
8
9  void new_tiles() {
10     int h, w;
11     cin >> h >> w;
12
13     vector<uint32_t> floor_plan(h, 0);
14     int p;
15     for (int i = 0; i < h; i++) {
16         for (int j = 0; j < w; j++) {
17             cin >> p;
18             if (p == 1) floor_plan[i] |= 0x01<<j;
19         }
20     }
21
22
23 }
24
25
26 int main() {
27     ios_base::sync_with_stdio(false);
28     int T; cin >> T;
29     for (int t=0; t < T; t++){
30         new_tiles();
31     }
32 }
```

# Golden Eye

**Keywords**— Delaunay Triangulation, Disjoint Sets

```
1 #include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
2 #include <CGAL/Delaunay_triangulation_2.h>
3 #include <CGAL/Triangulation_vertex_base_with_info_2.h>
4 #include <CGAL/Triangulation_face_base_2.h>
5 #include <vector>
6 #include <boost/pending/disjoint_sets.hpp>
7
8 typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
9 typedef K::Point_2 P;
10 typedef CGAL::Triangulation_vertex_base_with_info_2<int,K> Vb;
11 typedef CGAL::Triangulation_face_base_2<K> Fb;
12 typedef CGAL::Triangulation_data_structure_2<Vb,Fb> Tds;
13 typedef CGAL::Delaunay_triangulation_2<K,Tds> Delaunay;
14 typedef Delaunay::Vertex_handle VertexH;
15 typedef Delaunay::Finite_edges_iterator EI;
16 typedef std::pair<P,int> IPoint;
17 typedef boost::disjoint_sets_with_storage<> Uf;
18
19 using namespace std;
20
21 struct Edge {
22     Edge(int u_, int v_, K::FT sql_) : u(u_), v(v_), sql(sql_) {}
23     int u, v; // endpoints
24     K::FT sql; // squared length
25 };
26
27 inline bool operator<(const Edge& e, const Edge& f) { return e.sql < f.sql; }
28
29 void goldeneye() {
30     int n, m; double p;
31     cin >> n >> m >> p;
32
33     // Read jammers and build Delaunay
34     vector<IPoint> jammers(n);
35     long x, y;
36     for (int i = 0; i < n; i++) {
37         cin >> x >> y;
38         jammers[i] = IPoint(P(x, y), i);
39     }
40     Delaunay t;
41     t.insert(jammers.begin(), jammers.end());
42
43     // Extract edges and sort by length
44     vector<Edge> edges;
45     edges.reserve(3*n);
46     for (EI e = t.finite_edges_begin(); e != t.finite_edges_end(); e++) {
47         edges.push_back(Edge(e->first->vertex((e->second+1)%3)->info(),
48                             e->first->vertex((e->second+2)%3)->info(),
49                             t.segment(e).squared_length()));
50     }
51     sort(edges.begin(), edges.end());
52
53     // Compute components with power consumption p
54     Uf ufp(n);
55     typedef vector<Edge>::const_iterator ECI;
56     for (ECI e = edges.begin(); e != edges.end() && e->sql <= p; e++)
57         ufp.union_set(e->u, e->v);
58
59     // Handle missions
60     K::FT a = 0, b = 0;
61     Uf ufa(n), ufb(n);
62     ECI ai = edges.begin(), bi = edges.begin();
63     int x0, y0, x1, y1;
64     for (int i = 0; i < m; i++) {
65         cin >> x0 >> y0 >> x1 >> y1;
66         P p0(x0, y0), p1(x1, y1);
67         VertexH v0 = t.nearest_vertex(p0), v1 = t.nearest_vertex(p1);
68         K::FT d = 4 * max(CGAL::squared_distance(p0, v0->point()),
69                         CGAL::squared_distance(p1, v1->point()));
70
71         if (d <= p && ufp.find_set(v0->info()) == ufp.find_set(v1->info())) {
72             // mission possible with power p -> also with b
73             cout << "y";
74             if (d > b) b = d;
75             for (; bi != edges.end() &&
76                 ufb.find_set(v0->info()) != ufb.find_set(v1->info()); bi++)
```

```

78         ufb.union_set(bi->u, bi->v);
79     } else cout << "n";
80     // ensure it is possible at power a
81     if (d > a) a = d;
82     for (; ai != edges.end() &&
83           ufa.find_set(v0->info()) != ufa.find_set(v1->info()); ai++)
84         ufa.union_set(ai->u, ai->v);
85
86 }
87 if (ai != edges.begin() && (ai-1)->sql > a) a = (ai-1)->sql;
88 if (bi != edges.begin() && (bi-1)->sql > b) b = (bi-1)->sql;
89 cout << endl << a << endl << b << endl;
90 }
91
92 int main() {
93     ios_base::sync_with_stdio(false);
94     cout << setiosflags(ios::fixed) << setprecision(0);
95     int T; cin >> T;
96     for (int t = 0; t < T; t++) {
97         goldeneye();
98     }
99 }

```

## Corbusier

*Keywords*—Dynamic Programming

```
1 #include <vector>
2 #include <iostream>
3
4 using namespace std;
5
6 void corbusier() {
7     int n, i, k;
8     cin >> n >> i >> k;
9
10    vector<int> disks(n);
11    vector<vector<bool>> > T(n, vector<bool>(k, false));
12
13    for (int j = 0; j < n; j++) cin >> disks[j];
14
15    int h;
16    for (int d = 0; d < n; d++) {
17        h = disks[d];
18
19        T[d][h%k] = true;
20        if (d > 0) {
21            for (int j = 0; j < k; j++) {
22                if (T[d-1][j]) {
23                    T[d][j] = true;
24                    T[d][(j+h)%k] = true;
25                }
26            }
27        }
28
29        if (T[d][i]) {
30            cout << "yes\n";
31            return;
32        }
33    }
34    cout << "no\n";
35 }
36
37
38 int main() {
39     ios_base::sync_with_stdio(false);
40     int T; cin >> T;
41     for (int t = 0; t < T; t++) {
42         corbusier();
43     }
44 }
```

## Placing Knights

**Keywords**— Maximum Cardinality Matching

```
1  #include <iostream>
2  #include <map>
3  #include <algorithm>
4  // BGL includes
5  #include <boost/graph/adjacency_list.hpp>
6  #include <boost/graph/max_cardinality_matching.hpp>
7  // Namespaces
8  using namespace std;
9  using namespace boost;
10
11
12
13 // BGL Graph definitions
14 typedef adjacency_list<vecS, vecS, undirectedS> Graph;
15 typedef graph_traits<Graph>::edge_descriptor Edge;
16 typedef graph_traits<Graph>::vertex_descriptor Vertex;
17
18
19 void add_connection(Graph& G, vector<vector<bool> >& chessboard, int n, int i, int j, int new_i,
20     int new_j) {
21     if (new_i >= n || new_i < 0) return;
22     if (new_j >= n || new_j < 0) return;
23     if ( !chessboard[new_i][new_j] ) return;
24
25     add_edge(i*n + j, new_i*n + new_j, G);
26     return;
27 }
28
29 void placing_knights() {
30     int n; cin >> n;
31     vector<vector<bool> > chessboard(n, vector<bool>(n));
32
33     // Read chessboardx
34     int c = 0;
35     for (int i = 0; i < n; i++) {
36         for (int j = 0; j < n; j++) {
37             bool p; cin >> p;
38             chessboard[i][j] = p;
39             if ( p ) c++;
40         }
41     }
42
43     Graph G(n*n);
44
45     for (int i = 0; i < n; i++) {
46         for (int j = 0; j < n; j++) {
47             if (chessboard[i][j]) {
48                 add_connection(G, chessboard, n, i, j, i-1, j-2);
49                 add_connection(G, chessboard, n, i, j, i-1, j+2);
50                 add_connection(G, chessboard, n, i, j, i+1, j-2);
51                 add_connection(G, chessboard, n, i, j, i+1, j+2);
52                 add_connection(G, chessboard, n, i, j, i-2, j-1);
53                 add_connection(G, chessboard, n, i, j, i-2, j+1);
54                 add_connection(G, chessboard, n, i, j, i+2, j-1);
55                 add_connection(G, chessboard, n, i, j, i+2, j+1);
56             }
57         }
58     }
59
60     vector<Vertex> mate(n*n);
61     checked_edmonds_maximum_cardinality_matching(G, &mate[0]);
62     int matches = matching_size(G, &mate[0]);
63
64     cout << c - matches << endl;
65 }
66
67 int main() {
68     ios_base::sync_with_stdio(false);
69     int T; cin >> T;
70     for (int t=0; t < T; t++){
71         placing_knights();
72     }
73 }
```



# Radiation

**Keywords**— Binary Search, LP/QP

```
1  #include <iostream>
2  #include <vector>
3  #include <cassert>
4  #include <cmath>
5  #include <CGAL/basic.h>
6  #include <CGAL/QP_models.h>
7  #include <CGAL/QP_functions.h>
8  #include <CGAL/Gmpz.h>
9
10 // choose exact integral type
11 typedef CGAL::Gmpz ET;
12
13 // program and solution types
14 typedef CGAL::Quadratic_program<long> Program;
15 typedef CGAL::Quadratic_program_solution<ET> Solution;
16
17 using namespace std;
18
19
20 bool solve(vector<vector<vector<long> > >& coefficients, const int h, const int t, const int d) {
21
22     int nb_coeff = coefficients[0][d].size();
23
24     Program lp (CGAL::SMALLER, false, 0, false, 0);
25     CGAL::Quadratic_program_options options;
26     options.set_pricing_strategy(CGAL::QP_BLAND); // Bland's rule
27
28     for (int i = 0; i < (h + t); i++) {
29         for (int k = 0; k < nb_coeff; k++) {
30             lp.set_a(k, i, coefficients[i][d][k]);
31         }
32         lp.set_b(i, 0);
33
34         if (i >= h) lp.set_a(nb_coeff, i, 1); // Epsilon
35     }
36     lp.set_c(nb_coeff, -1);
37     lp.set_l(nb_coeff, true, 0);
38
39     Solution s = CGAL::solve_linear_program(lp, ET(), options);
40     return s.is_unbounded();
41 }
42
43
44 void radiation(){
45     int h, t;
46     cin >> h >> t;
47
48     const int D = 30;
49
50     vector<vector<long> > health(h, vector<long>(3)), tumor(t, vector<long>(3));
51
52     for (int i = 0; i < h; i++) cin >> health[i][0] >> health[i][1] >> health[i][2];
53     for (int i = 0; i < t; i++) cin >> tumor[i][0] >> tumor[i][1] >> tumor[i][2];
54
55     vector<vector<vector<long> > > coefficients(h+t, vector<vector<long> >(D+1));
56
57     for (int i = 0; i < (h + t); i++) {
58         vector<long> cell = (i < h) ? health[i] : tumor[i-h];
59
60         for (int d = 1; d <= D; d++) {
61             long coef_x = 1;
62             for (int x = 0; x <= d; x++) {
63                 long coef_y = 1;
64                 for (int y = 0; y <= d; y++) {
65                     long coef_z = 1;
66                     for (int z = 0; z <= d; z++) {
67                         if (x + y + z > d) continue;
68                         if (i >= h) coefficients[i][d].push_back(coef_x * coef_y * coef_z);
69                         else coefficients[i][d].push_back(-coef_x * coef_y * coef_z);
70                         coef_z *= cell[2];
71                     }
72                     coef_y *= cell[1];
73                 }
74                 coef_x *= cell[0];
75             }
76         }
77     }
```

```

78
79     int upper = 1;
80     while (!solve(coefficients, h, t, upper) && upper <= D) upper *= 2;
81
82     if (upper > D) cout << "impossible" << endl;
83
84     int low = upper / 2;
85     while (low + 1 != upper) {
86         int mid = low + (upper - low) / 2;
87         if (solve(coefficients, h, t, mid)) {
88             upper = mid;
89         } else low = mid;
90     }
91
92     cout << upper << endl;
93 }
94
95
96 int main() {
97     int T; cin >> T;
98     for (int t = 0; t < T; t++) {
99         radiation();
100     }
101 }

```

# The Empire Strikes Back

**Keywords**— *Keywords*— Delaunay Triangulation, LP/QP

```
1  #include <vector>
2  #include <iostream>
3  #include <algorithm>
4  #include <limits.h>
5  #include <CGAL/basic.h>
6  #include <CGAL/QP_models.h>
7  #include <CGAL/QP_functions.h>
8  #include <CGAL/Gmpzf.h>
9  #include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
10 #include <CGAL/Delaunay_triangulation_2.h>
11
12 // LP
13 typedef CGAL::Gmpzf ET;
14 typedef CGAL::Quadratic_program<double> Program;
15 typedef CGAL::Quadratic_program_solution<ET> Solution;
16 // Delaunay
17 typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
18 typedef CGAL::Delaunay_triangulation_2<K> Triangulation;
19 typedef K::Point_2 Point;
20
21 using namespace std;
22
23 void strikes_back() {
24     int a, s, b, e;
25     cin >> a >> s >> b >> e;
26
27     vector<Point> asteroid(a), shooting_points(s), bounty_hunters(b);
28     vector<int> asteroid_densities(a);
29     long x, y;
30     // Read data
31     for (int i = 0; i < a; i++) {
32         cin >> x >> y;
33         asteroid[i] = Point(x, y);
34         cin >> asteroid_densities[i];
35     }
36     for (int i = 0; i < s; i++) {
37         cin >> x >> y;
38         shooting_points[i] = Point(x, y);
39     }
40     for (int i = 0; i < b; i++) {
41         cin >> x >> y;
42         bounty_hunters[i] = Point(x, y);
43     }
44
45     // Create a triangulation for the bounty hunters to find the closest one at each shooting
46     // point
47     Triangulation bh;
48     bh.insert(bounty_hunters.begin(), bounty_hunters.end());
49
50     // Initialize linear program
51     Program lp (CGAL::LARGER, true, 0, false, 0);
52
53     double max_d = LONG_MAX; // Distance to the closest bounty hunters
54     double d;
55     for (int i = 0; i < s; i++) {
56         if (b > 0) { // In case the set of bounty hunters are empty
57             Point cl_bh = bh.nearest_vertex(shooting_points[i])->point();
58             max_d = CGAL::to_double(CGAL::squared_distance(cl_bh, shooting_points[i]));
59         }
60
61         for (int j = 0; j < a; j++) {
62             // Compute the distance of every shooting point to each asteroid
63             d = CGAL::to_double(CGAL::squared_distance(shooting_points[i], asteroid[j]));
64             // If the closests bounty hunter is nearer than the asteroid, then to this no energy
65             // will arrive
66             if (d >= max_d) continue;
67
68             // If arrives compute the coefficient
69             double coef = 1 / ((double) max(1., d));
70             lp.set_a(i, j, coef);
71         }
72         lp.set_c(i, 1); // Minimize the sum of energy at each shot
73         lp.set_a(i, a, 1); // (*) Set a last inequality upbounding the total amount of energy
74         // used
75     }
76     lp.set_b(a, e); // (*)
77     lp.set_r(a, CGAL::SMALLER); // (*)
```

```

77 // Set a lower limit to the amount of energy that must arrive to each asteroid
78 for (int j = 0; j < a; j++) {
79     lp.set_b(j, asteroid_densities[j]);
80 }
81
82 // Solve LP
83 Solution sol = CGAL::solve_linear_program(lp, ET());
84 assert (sol.solves_linear_program(lp));
85
86 // If exists solution then its possible to do it
87 if (sol.is_optimal()) cout << "y" << endl;
88 else cout << "n" << endl;
89 }
90
91 int main() {
92     int T; cin >> T;
93     for (int i = 0; i < T; i++) {
94         strikes_back();
95     }
96 }
97

```

## Bob's Burden

*Keywords*— Dijkstra Shortest Path

```
1 // Includes
2 // =====
3 // STL includes
4 #include <iostream>
5 #include <vector>
6
7 // BGL includes
8 #include <boost/graph/adjacency_list.hpp>
9 #include <boost/graph/dijkstra_shortest_paths.hpp>
10 // Namespaces
11 using namespace std;
12 using namespace boost;
13
14 // BGL Graph definitions
15 // =====
16 // Graph Type, OutEdgeList Type, VertexList Type, (un)directedS
17 typedef adjacency_list<vecS, vecS, directedS, // Use vecS for the VertexList! Choosing setS
18     for the OutEdgeList disallows parallel edges.
19     no_property, // interior properties of vertices
20     property<edge_weight_t, int> // interior properties of edges
21     > Graph;
22 typedef graph_traits<Graph>::edge_descriptor Edge; // Edge Descriptor: an object that
23     represents a single edge.
24 typedef graph_traits<Graph>::vertex_descriptor Vertex; // Vertex Descriptor: with vecS
25     vertex list, this is really just an int in the range [0, num_vertices(G)).
26 typedef graph_traits<Graph>::edge_iterator EdgeIt; // to iterate over all edges
27 typedef graph_traits<Graph>::out_edge_iterator OutEdgeIt; // to iterate over all outgoing
28     edges of a vertex
29 typedef property_map<Graph, edge_weight_t>::type WeightMap; // property map to access the
30     interior property edge_weight_t
31
32 void bobs_burden() {
33     int k; cin >> k;
34
35     int N = k * (k+1) / 2;
36     Graph G(2 * N);
37     WeightMap weightmap = get(edge_weight, G);
38
39     int pos_in, pos_out, v;
40     Edge e; bool success;
41     for (int i = 0; i < k; i++) {
42         for (int j = 0; j <= i; j++) {
43             pos_in = i * (i + 1) / 2 + j;
44             pos_out = pos_in + N;
45             cin >> v;
46             tie(e, success) = add_edge(pos_in, pos_out, G);
47             weightmap[e] = v;
48             if (j > 0) {
49                 int pos_left_in = pos_in - 1, pos_left_out = pos_out - 1;
50                 add_edge(pos_left_out, pos_in, G);
51                 add_edge(pos_out, pos_left_in, G);
52             }
53             if (i < k - 1) {
54                 int pos_down_left_in = pos_in + (i + 1);
55                 int pos_down_left_out = pos_down_left_in + N;
56                 int pos_down_right_in = pos_down_left_in + 1;
57                 int pos_down_right_out = pos_down_right_in + N;
58                 add_edge(pos_out, pos_down_left_in, G);
59                 add_edge(pos_down_left_out, pos_in, G);
60                 add_edge(pos_out, pos_down_right_in, G);
61                 add_edge(pos_down_right_out, pos_in, G);
62             }
63         }
64     }
65
66     vector<vector<int>> distmap(3, vector<int>(N*2));
67     // Compute the distance map for the three point (0,0), (k,0), (k,k)
68     dijkstra_shortest_paths(G, N,
69         distance_map(make_iterator_property_map(distmap[0].begin(), get(vertex_index, G))));
70     dijkstra_shortest_paths(G, 2*N - k,
71         distance_map(make_iterator_property_map(distmap[1].begin(), get(vertex_index, G))));
72     dijkstra_shortest_paths(G, 2*N - 1,
73         distance_map(make_iterator_property_map(distmap[2].begin(), get(vertex_index, G))));
74
75     int minimum_burden = INT_MAX;
76     for (int i = 0; i < k; i++) {
```

```

73     for (int j = 0; j <= i; j++) {
74         pos_in = i * (i + 1) / 2 + j;
75         if (pos_in == 0 || pos_in == N - k || pos_in == N - 1)
76             continue;
77         tie(e, success) = edge(pos_in, pos_in+N, G);
78         minimum_burden = min(minimum_burden,
79                               distmap[0][pos_in] + distmap[1][pos_in] + distmap[2][pos_in] + weightmap[e]);
80     }
81 }
82
83 cout << minimum_burden << endl;
84
85 }
86
87 int main() {
88     ios_base::sync_with_stdio(false);
89     int T; cin >> T;
90     for (int t=0; t < T; t++){
91         bobs_burden();
92     }
93 }

```

```

1  #include <vector>
2  #include <climits>
3  #include <iostream>
4  #include <algorithm>
5
6  using namespace std;
7
8
9  int dp(vector<vector<int> >& sum, int pos_a, int pos_b, vector<vector<int> >& memory) {
10
11     // Check memory first
12     if (memory[pos_a][pos_b] != -1) return memory[pos_a][pos_b];
13
14     // Look for the best combination to reduce the cost given an specific position of the two
15     // stacks
16     int pick_a, pick_b, new_cost, cost = INT_MAX;
17
18     // Separate the iteration, first look at the first stack and take the first volume
19     pick_a = sum[0][pos_a] - sum[0][pos_a-1];
20     for(int i = pos_b - 1; i >= 0; i--) {
21         if ((pos_a-1 == 0 && i > 0) || (pos_a-1 > 0 && i == 0)) continue;
22         pick_b = sum[1][pos_b] - sum[1][i];
23
24         new_cost = (memory[pos_a-1][i] != -1) ? memory[pos_a-1][i] : dp(sum, pos_a-1, i, memory);
25         cost = min(cost, new_cost + pick_a*pick_b);
26     }
27
28     pick_b = sum[1][pos_b] - sum[1][pos_b-1];
29     for(int i = pos_a - 1; i >= 0; i--) {
30         if ((i == 0 && pos_b-1 > 0) || (i > 0 && pos_b-1 == 0)) continue;
31         pick_a = sum[0][pos_a] - sum[0][i];
32
33         new_cost = (memory[i][pos_b-1] != -1) ? memory[i][pos_b-1] : dp(sum, i, pos_b-1, memory);
34         cost = min(cost, new_cost + pick_a*pick_b);
35     }
36
37     memory[pos_a][pos_b] = cost;
38     return cost;
39 }
40
41 void dhl() {
42     int n; cin >> n;
43
44     vector<vector<int> > sum(2, vector<int>(n+1));
45     sum[0][0] = 0; sum[1][0] = 0;
46
47     int v;
48     for (int i = 0; i < 2; i++) {
49         for (int j = 0; j < n; j++) {
50             cin >> v; v--;
51             sum[i][j+1] = sum[i][j] + v;
52         }
53     }
54
55     int pos_a = n, pos_b = n;
56
57     // Build memory matrix
58     vector<vector<int> > memory(n+1, vector<int>(n+1, -1));
59     memory[0][0] = 0;
60
61     int cost = dp(sum, pos_a, pos_b, memory);
62     cout << cost << endl;
63 }
64
65
66 int main() {
67     ios_base::sync_with_stdio(false);
68     int T; cin >> T;
69     for (int t=0; t < T; t++){
70         dhl();
71     }
72 }

```

## Sweepers

**Keywords**— Maximum Flow, Connected Components

```
1
2 #include <iostream>
3 #include <map>
4 #include <algorithm>
5 // BGL includes
6 #include <boost/graph/adjacency_list.hpp>
7 #include <boost/graph/push_relabel_max_flow.hpp>
8 #include <boost/graph/connected_components.hpp>
9 // Namespaces
10 using namespace std;
11 using namespace boost;
12
13
14 // BGL Graph definitions
15 // =====
16 // Graph Type with nested interior edge properties for Flow Algorithms
17 typedef adjacency_list_traits<vecS, vecS, directedS> Traits;
18 typedef adjacency_list<vecS, vecS, directedS, no_property,
19     property<edge_capacity_t, long,
20     property<edge_residual_capacity_t, long,
21     property<edge_reverse_t, Traits::edge_descriptor> > > > Graph;
22 // Interior Property Maps
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;
29
30
31 // Custom Edge Adder Class, that holds the references
32 // to the graph, capacity map and reverse edge map
33 // =====
34 class EdgeAdder {
35     Graph &G;
36     EdgeCapacityMap &capacitymap;
37     ReverseEdgeMap &revedgemap;
38
39 public:
40     // to initialize the Object
41     EdgeAdder(Graph & G, EdgeCapacityMap &capacitymap, ReverseEdgeMap &revedgemap):
42         G(G), capacitymap(capacitymap), revedgemap(revedgemap){}
43
44     // to use the Function (add an edge)
45     Edge addEdge(int from, int to, long capacity) {
46         Edge e, reverseE;
47         bool success;
48         tie(e, success) = add_edge(from, to, G);
49         tie(reverseE, success) = add_edge(to, from, G);
50         capacitymap[e] = capacity;
51         capacitymap[reverseE] = 0;
52         revedgemap[e] = reverseE;
53         revedgemap[reverseE] = e;
54         return e;
55     }
56 };
57
58
59 void sweepers() {
60     int n, m, s, p;
61     cin >> n >> m >> s;
62
63     Graph G(n+2);
64     EdgeCapacityMap capacitymap = get(edge_capacity, G);
65     ReverseEdgeMap revedgemap = get(edge_reverse, G);
66     EdgeAdder eaG(G, capacitymap, revedgemap);
67     Vertex v_source = n, v_target = n + 1;
68
69     vector<int> degree(n, 0);
70     vector<bool> has_edges(n, false);
71     has_edges[n] = true; has_edges[n+1];
72
73     for (int i = 0; i < s; i++) {
74         cin >> p;
75         eaG.addEdge(v_source, p, 1);
76         degree[p]++;
77         has_edges[p] = true;
```



```

78     }
79     for (int i = 0; i < s; i++) {
80         cin >> p;
81         eaG.addEdge(p, v_target, 1);
82         degree[p]++;
83         has_edges[p] = true;
84     }
85
86     int x, y;
87     for (int i = 0; i < m; i++) {
88         cin >> x >> y;
89         eaG.addEdge(x, y, 1);
90         eaG.addEdge(y, x, 1);
91         degree[x]++; degree[y]++;
92         has_edges[x] = true; has_edges[y] = true;
93     }
94
95     int flow = push_relabel_max_flow(G, v_source, v_target);
96
97     vector<Vertex> componentmap(n+2);
98     connected_components(G, make_iterator_property_map(componentmap.begin(), get(vertex_index, G))
99         );
100
101     bool result = flow == s;          // First check if the flow is equal the number of sweepers
102     // Now check the degree of all the nodes
103     for (int i = 0; i < n; i++)
104         if (degree[i] % 2 != 0) result = false;
105     // Now check that all the nodes from the same component that source, are the ones that have
106     // the
107     // edges -> the edges are reachable.
108     int ref_component = componentmap[n];
109     for (int i = 0; i < n; i++) {
110         if (has_edges[i] && componentmap[i] != ref_component) result = false;
111     }
112
113     if (result) cout << "yes" << endl;
114     else cout << "no" << endl;
115 }
116
117 int main() {
118     ios_base::sync_with_stdio(false);
119     int T; cin >> T;
120     for (int t=0; t < T; t++){
121         sweepers();
122     }
123 }

```

## Portfolios revisited

### Keywords— LP/QP

```
1 #include <iostream>
2 #include <cassert>
3 #include <cmath>
4 #include <CGAL/basic.h>
5 #include <CGAL/QP_models.h>
6 #include <CGAL/QP_functions.h>
7 #include <CGAL/Gmpz.h>
8
9 // choose exact integral type
10 typedef CGAL::Gmpz ET;
11
12 // program and solution types
13 typedef CGAL::Quadratic_program<int> Program;
14 typedef CGAL::Quadratic_program_solution<ET> Solution;
15
16 using namespace std;
17
18
19 void portfolios(int n, int m){
20
21     // Quadratic Programming
22     Program qp(CGAL::SMALLER, true, 0, false, 0);
23     const int COST = 0;
24     const int RETURN = 1;
25     int c, r, v;
26     vector<int> returns(n);
27
28     qp.set_r(RETURN, CGAL::LARGER);
29     for (int i = 0; i < n; i++) {
30         cin >> c >> r;
31         qp.set_a(i, COST, c);
32         qp.set_a(i, RETURN, r);
33         returns[i] = r;
34     }
35     for (int i = 0; i < n; i++) {
36         for (int j = 0; j < n; j++) {
37             cin >> v;
38             if (j <= i) qp.set_d(i, j, 2*v);
39         }
40     }
41     for (int i = 0; i < m; i++) {
42         long C, V;
43         cin >> C >> V;
44
45         qp.set_b(COST, C);
46
47         int lower = 0, upper = INT_MAX, middle;
48         double best_return = 0;
49         // Binary search
50         while (lower + 1 != upper) {
51             middle = lower + (upper - lower) / 2;
52             qp.set_b(RETURN, middle);
53
54             Solution s = CGAL::solve_nonnegative_quadratic_program(qp, ET());
55             if (s.is_optimal() && CGAL::to_double(s.objective_value()) <= V) {
56                 lower = middle;
57                 // Compute return
58                 double total_return = 0;
59                 int indx = 0;
60                 for (auto it = s.variable_values_begin(); it != s.variable_values_end(); it++) {
61                     total_return += returns[indx++] * CGAL::to_double(*it);
62                 }
63                 best_return = max(best_return, total_return);
64             } else {
65                 upper = middle;
66             }
67         }
68         cout << best_return << endl;
69     }
70 }
71
72 }
73
74
75 int main() {
76     ios_base::sync_with_stdio(false);
77     int n, m;
```

```
78     while (true) {  
79         cin >> n >> m;  
80         if (n == 0 && m == 0) { break; }  
81         portfolios(n, m);  
82     }  
83 }
```

# The Phantom Menace

*Keywords*— Maximum Flow

```
1  /*****
2  Given a directed graph find how many vertex need to be removed to disconnect one set of vertex with
3  another set.
4  This is solve creating a graph with an income and outcome node for each vertex. All the edges must
5  have capacity one and then finding the mincut of the graph is possible to find the number of
   vertex
6  required to disconnect the two sets of vertex. The mincut is found computing the maximum flow.
7  *****/
8
9  // Includes
10 // =====
11 // STL includes
12 #include <iostream>
13 #include <cstdlib>
14 // BGL includes
15 #include <boost/graph/adjacency_list.hpp>
16 #include <boost/graph/push_relabel_max_flow.hpp>
17 // Namespaces
18 using namespace boost;
19 using namespace std;
20
21 // BGL Graph definitions
22 // =====
23 // Graph Type with nested interior edge properties for Cost Flow Algorithms
24 typedef adjacency_list_traits<vecS, vecS, directedS> Traits;
25 typedef adjacency_list<vecS, vecS, directedS, no_property,
26     property<edge_capacity_t, long,
27     property<edge_residual_capacity_t, long,
28     property<edge_reverse_t, Traits::edge_descriptor> > > > Graph;
29 // Interior Property Maps
30 typedef property_map<Graph, edge_capacity_t>::type      EdgeCapacityMap;
31 typedef property_map<Graph, edge_residual_capacity_t>::type ResidualCapacityMap;
32 typedef property_map<Graph, edge_reverse_t>::type      ReverseEdgeMap;
33 typedef graph_traits<Graph>::vertex_descriptor        Vertex;
34 typedef graph_traits<Graph>::edge_descriptor          Edge;
35 typedef graph_traits<Graph>::out_edge_iterator         OutEdgeIt;
36
37 // Custom Edge Adder Class, that holds the references
38 // to the graph, capacity map, weight map and reverse edge map
39 // =====
40 class EdgeAdder {
41     Graph &G;
42     EdgeCapacityMap &capacitymap;
43     ReverseEdgeMap &revedgemap;
44
45 public:
46     // to initialize the Object
47     EdgeAdder(Graph & G, EdgeCapacityMap &capacitymap, ReverseEdgeMap &revedgemap):
48         G(G), capacitymap(capacitymap), revedgemap(revedgemap){}
49
50     // to use the Function (add an edge)
51     void addEdge(int from, int to, long capacity) {
52         Edge e, reverseE;
53         bool success;
54         tie(e, success) = add_edge(from, to, G);
55         tie(reverseE, success) = add_edge(to, from, G);
56         capacitymap[e] = capacity;
57         capacitymap[reverseE] = 0;
58         revedgemap[e] = reverseE;
59         revedgemap[reverseE] = e;
60     }
61 };
62
63
64 void phantom_menace() {
65     int n, m, s, d;
66     cin >> n >> m >> s >> d;
67
68     // Create Graph and Maps
69     Graph G(2*n+2);      // Input nodes: [0, n)  Output nodes: [n, 2*n)
70     EdgeCapacityMap capacitymap = get(edge_capacity, G);
71     ReverseEdgeMap revedgemap = get(edge_reverse, G);
72     EdgeAdder eaG(G, capacitymap, revedgemap);
73
74     Vertex src = 2 * n, sink = 2 * n + 1;
75
76     // Reading graphs edges
```

```

77     int i, j;
78     for (int k = 0; k < m; k++) {
79         cin >> i >> j;
80         eaG.addEdge(n + i, j, 1);
81     }
82     for (int k = 0; k < n; k++) eaG.addEdge(k, n + k, 1);
83     // Read starting vertex
84     for (int k = 0; k < s; k++) {
85         cin >> i;
86         eaG.addEdge(src, i, 1);
87     }
88     // Read ending vertex
89     for (int k = 0; k < d; k++) {
90         cin >> j;
91         eaG.addEdge(n + j, sink, 1);
92     }
93     // Compute the min cut as max flow
94     int flow = push_relabel_max_flow(G, src, sink);
95     cout << flow << endl;
96 }
97
98
99
100 int main() {
101     int T; cin >> T;
102     for (int i = 0; i < T; i++) {
103         phantom_menace();
104     }
105 }

```

## Cantonal Courier

*Keywords*— Maximum Flow

```
1 #include <vector>
2 #include <iostream>
3 // BGL includes
4 #include <boost/graph/adjacency_list.hpp>
5 #include <boost/graph/push_relabel_max_flow.hpp>
6 // Namespaces
7 using namespace std;
8 using namespace boost;
9
10
11 // BGL Graph definitions
12 // =====
13 // Graph Type with nested interior edge properties for Flow Algorithms
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> > > > Graph;
19 // Interior Property Maps
20 typedef property_map<Graph, edge_capacity_t>::type      EdgeCapacityMap;
21 typedef property_map<Graph, edge_residual_capacity_t>::type ResidualCapacityMap;
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
26
27 // Custom Edge Adder Class, that holds the references
28 // to the graph, capacity map and reverse edge map
29 // =====
30 class EdgeAdder {
31     Graph &G;
32     EdgeCapacityMap &capacitymap;
33     ReverseEdgeMap &revedgemap;
34
35 public:
36     // to initialize the Object
37     EdgeAdder(Graph & G, EdgeCapacityMap &capacitymap, ReverseEdgeMap &revedgemap):
38         G(G), capacitymap(capacitymap), revedgemap(revedgemap){}
39
40     // to use the Function (add an edge)
41     void addEdge(int from, int to, long capacity) {
42         Edge e, reverseE;
43         bool success;
44         tie(e, success) = add_edge(from, to, G);
45         tie(reverseE, success) = add_edge(to, from, G);
46         capacitymap[e] = capacity;
47         capacitymap[reverseE] = 0;
48         revedgemap[e] = reverseE;
49         revedgemap[reverseE] = e;
50     }
51 };
52
53 void courier() {
54     int Z, J; cin >> Z >> J;
55
56     Graph G(Z + J);
57     EdgeCapacityMap capacitymap = get(edge_capacity, G);
58     ReverseEdgeMap revedgemap = get(edge_reverse, G);
59     // ResidualCapacityMap rescapacitymap = get(edge_residual_capacity, G);
60     EdgeAdder eaG(G, capacitymap, revedgemap);
61
62     Vertex src = add_vertex(G), sink = add_vertex(G);
63
64     // Read zone costs
65     int c;
66     for (int i = 0; i < Z; i++) {
67         cin >> c;
68         eaG.addEdge(J+i, sink, c);
69     }
70
71     // Read jobs profits
72     int total_revenue = 0;
73     for (int i = 0; i < J; i++) {
74         cin >> c;
75         eaG.addEdge(src, i, c);
76         total_revenue += c;
77     }
```

```

78
79 // Read jobs and zones connections
80 for (int i = 0; i < J; i++) {
81     int N; cin >> N;
82     for (int j = 0; j < N; j++) {
83         int z; cin >> z;
84         eaG.addEdge(i, J+z, INT_MAX);
85     }
86 }
87
88 long flow = push_relabel_max_flow(G, src, sink);
89
90 cout << total_revenue - flow << endl;
91 }
92
93 int main() {
94     ios_base::sync_with_stdio(false);
95     int T; cin >> T;
96     for (int t = 0; t < T; t++) {
97         courier();
98     }
99 }

```

# Index

- Backtracking, 91
- Binary Search, 13, 32, 84, 97
- Delaunay Triangulation, 61–63, 93, 99
  - Nearest Neighbor, 65, 81, 84, 86
- DFS, 13, 63, 71, 73
- Disjoint Sets, 93
- Dynamic Programming, 34, 36, 39, 43, 51, 95, 103
  - Split and List, 37, 91
- Geometry
  - Intersection, 16, 17
  - Minimum Circle, 21, 32
  - Triangle Contains, 19
- Graphs
  - Biconnected Components, 28
  - Bipartite Matchings, 71
  - Connected Components, 84, 104
  - Dijkstra Shortest Path, 24, 26, 41, 79, 101
  - Max Flow Min Cost, 69, 75, 77, 82, 89
  - Max Flow Min Cut, 73
  - Maximum Cardinality Matching, 30, 41, 96
  - Maximum Flow, 45, 47, 49, 59, 71, 104, 108, 110
  - Minimum Spanning Tree, 24, 26
- Greedy, 12, 15
  - Interval Scheduling, 11, 22
- LP/QP, 53, 55, 56, 58, 67, 97, 99, 106
- Sliding Window, 9, 10, 19
- Tricks
  - Precompute, 4, 6
  - Tree Search, 7