Uh, lepog li zadatka – grafovi (priprema za pismeni)

Zadatak 1.
https://csacademy.com/contest/archive/task/x-distance/statement/

X Distance

Time limit: *1000 ms*
Memory limit: *128 MB*

You are given a weighted undirected graph with N nodes and M edges. We define the cost of a path to be equal to the maximum weight of an edge on the path. Find the number of pairs of nodes for which the minimum cost of a path between them is equal to X.

Rešenje (komisijsko (Velea) u odnosu na koje su postavljena ograničenja)

Ideja:
1. Nađite broj parova čije rastojenje nije veće od X
2. Oduzmite broj parova čije rastojanje je najviše $X-1$.

Realizacija koraka 1
Uklonimo sve grane teže od X. Nađimo parove čvorova (i njihove komponenete povezanosti) između kojih postoje putevi čija cena je najviše X.
Izbrojmo broj parova u komponentama.

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

typedef long long int64;

const int kMaxN = 1e5+5;

vector<pair<int, int>> vertex[kMaxN];

int64 Solve(int n, int cost) {
  int64 ans = 0;
  vector<bool> visited(n + 1, 0);

  for (int i = 1; i <= n; i += 1) {
    if (visited[i]) {
      continue;
    }

    vector<int> queue = {i};
    visited[i] = true;
```

```
      int q_size = 1;

      while (queue.size()) {
         int node = queue.back();
         queue.pop_back();

         for (auto edge : vertex[node]) {
            if (edge.second > cost) {
               continue;
            }

            int oth = edge.first;
            if (visited[oth]) {
               continue;
            } else {
               queue.push_back(oth);
               visited[oth] = true;
               q_size += 1;
            }
         }
      }

      ans += 1LL * q_size * (q_size - 1) / 2;
   }

   return ans;
}

int main() {
   int n, m, x;
   cin >> n >> m >> x;
   for (int i = 0; i < m; i += 1) {
      int a, b, c;
      cin >> a >> b >> c;
      vertex[a].push_back({b, c});
      vertex[b].push_back({a, c});
   }

   int64 small = Solve(n, x - 1);
   int64 big = Solve(n, x);

   cout << big - small << '\n';

   return 0;
}
```

Zadatak 2

https://csacademy.com/contest/archive/task/manhattan-distances/statement/

Manhattan Distances

Time limit: *1000 ms*
Memory limit: *256 MB*

Alex drew 3 points at **integer** coordinates, but then he lost the paper. Now he only remembers the Manhattan distance between every pair of points. Help him find a possible set of 3 points that respect these distances.

Standard input

The first line contains a single integer T representing the number of test cases that follow.

Each of the next T lines contains 3 integer, representing the Manhattan distances between the points. Note that these numbers are not given in any particular order.

Standard output

For each test print the answer on a single line.

If there is no solution, output -1. Otherwise, print 6 integers x_1,y_1,x_2,y_2,x_3,y_3 representing the coordinates of the 3 points.

Constraints and notes

- $1 \leq T \leq 10000$
- The distances are integers between 1 and 10^8
- You can print any solution where the coordinate of the points are integers between -10^8 and 10^8

Ideja

Let's analyse the cases when there is no solution. First, the triangle inequality should still hold: the largest distance should not exceed the sum of the other two.

The other necessary condition is for the sum of the three distances to be an even number. Here's why: if we have three points and their x-coordinates are $x_1 < x_2 < x_3$, they will contribute to the total sum by $2 * (x_3 - x_1)$. The same reasoning applies for the y-coordinates.

In all the other cases, we have a solution. There are many possible ways to do this. In our solution, we choose $(0,0)$, $(d_1,0)$ and $\left(\frac{d_2-d_3+d_1}{2}, d_2 - \frac{d_2-d_3+d_1}{2}\right)$

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

```
vector<pair<int, int>> solve(vector<int> d) {
   int maxDist = *max_element(d.begin(), d.end());
   int sum = accumulate(d.begin(), d.end(), 0);
   if (2 * maxDist > sum || sum % 2 == 1) {
      return vector<pair<int, int>>();
   }

   vector<pair<int, int>> points;
   points.push_back(make_pair(0, 0));
   points.push_back(make_pair(d[0], 0));
   int x = (d[1] - d[2] + d[0]) / 2;
   int y = d[1] - x;
   points.push_back(make_pair(x, y));
   return points;
}


int main() {
   int t; cin >> t;
   for (int test = 0; test < t; ++test) {
      vector<int> d(3);
      for (auto &it : d) {
         cin >> it;
      }

      vector<pair<int, int>> points = solve(d);
      if (points.size()) {
         for (auto it : points) {
            cout << it.first << " " << it.second << " ";
         }
      } else {
         cout << "-1";
      }
      cout << "\n";
   }
   return 0;
}
```

Zadatak 3

https://csacademy.com/contest/archive/task/spanning-trees/statement/

Spanning Trees

Time limit: *1000 ms*
Memory limit: *256 MB*

You are given two integers N and K. Generate a weighted undirected graph with N nodes such that:

- Both the minimum and the maximum spanning trees are unique
- The minimum and the maximum spanning trees have exactly K edges in common

Standard input

The first line contains two integers N and K.

Standard output

If there is no solution, output -1.

Otherwise, on the first line print a single integer M, the number of edges of the graph.

Each of the next M line should contain three integers a,b,c, representing an edge between nodes a and b with cost c.

Constraints and notes

- $0 \leq K < N \leq 10^5$
- Multiple edges or self-loops are not allowed
- The costs of the edges should be between 1 and 10^9
- M should be at most $2 * N$

Rešenje (komisijsko (Velea) u odnosu na koje su postavljena ograničenja)

Solution

This is a constructive problem that admits more than one solution.

First, we need to identify the cases when there is no solution:

- $N=2$ and $K=0$, because there is only 1 edge that is going to be shared by both spanning trees
- $N=3$ and $K=0$, because there are at most 3 edges, the spanning trees have 2 edges, so they will share at least 1 edge

In all the other cases, there is a solution.

An interesting observation is that if we have a solution for a pair (N,K), we also have a solution for $(N+1,K+1)$: we just create an external node. So we just need to find a solution for pairs of the form $(N-K,0)$ and then adding external nodes. One possible way to do this is:

- edges of cost 1 for $1-2-\ldots-N$
- edges of cost 10^9 for 2-4-6...-1-3-5...

```
#include <bits/stdc++.h>
using namespace std;
int main() {
   ios::sync_with_stdio(false);
   cin.tie(0);
   int n, k;
   cin >> n >> k;
   if (k == 0 && (n == 2 || n == 3)) {
      cout << -1 << '\n';
      return 0;
   }
   int edges = 0, vertices = 1;
   int d = n - k;
   cout << n + d - 2 << '\n';
   if (d == 2) {
      cout << 1 << ' ' << 2 << ' ' << ++edges << '\n';
      cout << 1 << ' ' << 3 << ' ' << ++edges << '\n';
      cout << 2 << ' ' << 3 << ' ' << ++edges << '\n';
      vertices = 3;
   }
   if (d == 3) {
      cout << 1 << ' ' << 2 << ' ' << ++edges << '\n';
      cout << 1 << ' ' << 3 << ' ' << ++edges << '\n';
      cout << 1 << ' ' << 4 << ' ' << ++edges << '\n';
      cout << 2 << ' ' << 4 << ' ' << ++edges << '\n';
      cout << 3 << ' ' << 4 << ' ' << ++edges << '\n';
      vertices = 4;
   }
   if (d >= 4) {
      for (int i = 2; i <= d; ++i)
         cout << i - 1 << ' ' << i << ' ' << ++edges << '\n';
      for (int i = 4; i <= d; i += 2)
         cout << i - 2 << ' ' << i << ' ' << ++edges << '\n';
      cout << d - d % 2 << ' ' << 1 << ' ' << ++edges << '\n';
      for (int i = 3; i <= d; i += 2)
         cout << i - 2 << ' ' << i << ' ' << ++edges << '\n';
      vertices = d;
   }
   while (vertices < n)
      cout << 1 << ' ' << ++vertices << ' ' << ++edges << '\n';
}
```