1. Brute Force TSP

tl;dr

Read the short description at the bottom.

Long description:

Fito is trying to go over all ($n = 100$) landmarks in Havana, but he wants to visit all of them using the shortest possible trip. Fito can start at any landmark and can visit them in whatever order he wants, but it is important that he doesn't visit any landmark twice, as this might be very boring.

Havana is a strange place so it is possible that $distance(1, 2) + distance(2, 3) > distance(1, 3)$. (Don't count with triangle inequality)

Since he is studying Computer Science, and he recently learnt about Traveling Salesman Problem (`https://en.wikipedia.org/wiki/Travelling_salesman_problem`) he made a program to find the shortest route among all landmarks. His code enumerated all $n!$ permutations in lexicographical order (`https://en.wikipedia.org/wiki/Permutation#Generation_in_lexicographic_order`), and check for the distance of going through all landmarks in that order. Whenever his code found a better path than all previously found paths he printed the distance of that path and the path itself.

He has a very very very very powerful computer, but yet he noticed after an hour that his code was not ending any time soon, so he decided to stop it and use the best path found so far. He stop it after checking the first ($k = 10^{17} = 100000000000000000$) permutations.

Maria wants to know how much distance Fito is walking, can you help her finding that distance.

All landmarks in Havana are enumerated from 0 to 99. The distance between a landmark and itself is always 0 and the distance between landmark U and landmark V is the same as the distance from V to U (for every possible pair). Fortunately the distance between all landmarks can be generated using the following c++ code:

```
// This is a matrix of 100 x 100 initialized with 0 at all positions.
int graph[100][100];

long long seed = 2122071420;
long long mult = 1103515245;
long long inc = 12345;
// This is the same as 2^31 - 1 (two to the power of thirty-one minus 1)
long long mod = (1LL << 31) - 1;

long long next_rand()
{
    return seed = (seed * mult + inc) & mod;
}

// Generate the graph
void generate(int n)
{
    long long checksum = 0;

    for (int i = 0; i < n; ++i)
    {
        for (int j = i + 1; j < n; ++j)
        {
            graph[j][i] = next_rand() % 999999 + 1;
            graph[i][j] = graph[j][i];
            checksum = (checksum + graph[i][j]) % 1000000007;
```

```
        }
    }

    cout << "checksum: " << checksum << endl;
}

int main() {
    generate(100);
}
```

The checksum is given to you just to check that you generate the graph correctly, it is not relevant in any other way.

```
checksum: 508586144
```

Short description:

Given a graph of ($n = 100$) nodes find the shortest route that pass through each node exactly once. Each permutation of the number from 0 to 99 defines a possible route to follow, and have associated the distance of the route following nodes in that order. The answer to this problem is the shortest distance among the first ($k = 10^{17}$) permutations if all the 100! (100 factorial) permutations are sorted lexicographically.