Code Jam 2017 - World Finals

Analysis: Teleporters

Teleporters: Analysis

This problem starts off with a strange twist, because instead of the regular Euclidean geometry (also known as L2 geometry), it uses L1 geometry. The reason, if you are curious, is that the distance between two points with integer coordinates is also an integer. A Euclidean geometry version of this had serious precision issues, and L1 geometry has all the properties that are needed. A concept that will come up later is the set of points that are at a particular distance from a center. In Euclidean geometry, that's just a sphere. In L1 geometry, though, is an octahedron with diagonals (segments that connect opposite vertices, passing through the center) parallel to the axis. Luckily, most intuitive properties of spheres in Euclidean geometry also work for spheres in L1 geometry, with the important exception of rotations, that are not used in this problem. If it helps you to visualize, for remainder of the text, you can think of the problem as working in L2 geometry. However, throughout the rest of this analysis, every time we say "distance", we are referring to L1 distance; we will write dist(x, y) for the L1 distance between points x and y. Every time we say "sphere", we are referring to L1 spheres (i.e., regular octahedra).

Small dataset

Let us start slowly: after 0 teleportations starting from a point P, the only reachable place is point P. After 1 teleportation, reachable places depend on which teleporter we used. If we use teleporter t, reachable points are the surface of the sphere with center t and radius dist(P, t). Let R_i be the set of points that are reachable in i teleportations using any set of teleporters. As we mentioned above, $R_0 = \{P\}$ and R_1 is a union of **N** sphere surfaces, one centered on each teleporter. What about R_2? If we use teleporter u for the second teleportation, we can land at any point that is at distance d from u, where d can take the value of any distance betwen u and a point in R_1. This implies R_2, and all other R_i, are also a union of sphere surfaces, possibly infinitely many.

Notice that R_1 is a connected continuous set because all the spheres' surfaces intersect at point P. Thus, the values for the distance d in the definition above form an interval, since the distance function is continuous. This implies that R_2 is actually a union of sphere differences: for each teleporter t, all points x such that $L_{t,2} \le dist(x, t) \le U_{t,2}$ are reachable. (Throughout these explanations, we use L to refer to a lower bound on a reachable range, and U for a corresponding upper bound.) Once again, all the sphere differences contain P, thus, they intersect and R_2 is a connected continuous set, which means the distances we use to calculate R_3 are intervals. This argument generalizes to prove by induction that each R_i is exactly the union over all teleporters t of all points x such that $L_{t,i} \le dist(x, t) \le U_{t,i}$.

This yields a dynamic programming algorithm that solves the Small dataset. Keep two arrays L and U representing the values $L_{t,i}$ and $U_{t,i}$ for a particular i and use them to calculate the next values $L_{t,i+1}$ and $U_{t,i+1}$. After each iteration, check whether $L_{t,i} \le dist(Q, t) \le U_{t,i}$ for some t, and if so, i is the answer.

By definition, $L_{t,i+1}$ and $U_{t,i+1}$ are the distances from t to its closest and farthest points in R_i, respectively. The farthest point in R_i from t is at a distance which is the maximum over all teleporters u of $dist(t, u) + U_{u,i}$ (this is the distance to the point on the surface of the sphere

centered at u with radius $U_{u,i}$ that is the opposite direction from t). The closest point is slightly more complicated. For each teleporter u we need to consider:

- dist(t, u) - $U_{u,i}$ if dist(t, u) > $U_{u,i}$ (t is outside the outer sphere centered at u),
- $L_{u,i}$ - dist(t, u) if dist(t, u) < $L_{u,i}$ (t is inside the inner sphere), or
- 0, in all other cases (t is in between, that is, it is itself a reachable point).

This means we can calculate each $L_{t,i}$ and $U_{t,i}$ in O(**N**) by comparing the values above for each teleporter u.

Notice that a point reachable in i teleportations is also reachable in i+1, i+2, ... etc teleportations, because you can use a teleporter to move from a point to itself. Thus, $U_{t,i}$ is non-decreasing with i, and $L_{t,i}$ is non-increasing with i. Additionally, since the distances dist(t, u) are positive, when **N** ≥ 2, the maximum over all t of $U_{t,i}$ is strictly increasing with i, and the minimum over all t of $L_{t,i}$ is strictly decreasing with i up to the first j where $L_{t,j} = 0$. That means, for **N** ≥ 2, the intervals grow until one of them represents a sphere covering the entire cube of values for Q within the limits. Moreover, since the values are integers, the increase and decrease is at least 1 per iteration, so the number of iterations needed to cover the entire region of valid Qs is bounded by 3M (M on each direction), where M is the number of valid coordinates, which is only 2001 in the Small dataset. This in particular means that for **N** ≥ 2 the answer is never impossible. For **N**=1, we can note that using the same teleporter twice in a row is never useful, so after 1 iteration, if Q is not reached, the answer is impossible.

The time complexity of the presented algorithm is O(M $\mathbf{N}^2$): up to 3M steps, each of which requires calculating O(**N**) values, and calculating each one requires an iteration over **N** other teleporters and constant-time math.

Large dataset

Of course, when M is bounded by $2 \times 10^{12} + 1$, a time complexity linear on M won't finish fast enough, so we have to do something else.

Let us focus first on deciding if it's possible to go from P to Q with a single teleportation. That means using a single teleporter t, and due to conservation of distance, it must be dist(P, t) = dist(Q, t). Moreover, this condition is sufficient and necessary for the answer to be 1. We can check for this condition initially with a loop over all teleporters in O(**N**) time.

As we saw on the Small, checking whether the answer is 1 is sufficient to fully solve cases with **N** = 1. We assume further that **N** ≥ 2.

Let us now consider the case where there exists two teleporters t and u such that dist(P, t) ≥ dist(Q, t) and dist(P, u) ≤ dist(Q, u). Consider the sphere A centered at t that passes through P, and the sphere B centered at u that passes through Q. By the assumed inequalities, A contains Q and B contains P, which means A and B intersect. Let x be any point at the intersection, for which dist(P, t) = dist(x, t) and dist(Q, u) = dist(x, u) hold. Then, x is a possible intermediate stop to go from P to Q in exactly 2 teleportations, so, if the inequalities hold, 2 is the answer. Notice there are other cases in which 2 is also the answer, which are covered below.

At this point, we can assume that either P is closer to any teleporter than Q, or vice versa (otherwise, we can choose two teleporters to fullfill the inequalities at the beginning of the previous paragraph). Since the problem is symmetric, swap P and Q if needed to make P the closest of P and Q to all teleporters.

Now recall the definitions of R, L and U from the Small solution. Since P is closest to all teleporters, dist(Q, t) > $U_{t,1}$ = dist(P, t) for all t. This means Q is outside the spheres centered in

all teleporters. Since $L_{t,i}$ is non-increasing with i, the inner sphere contracts with each step, which means Q is never inside the inner sphere, so as soon as Q is inside the outer sphere, we are guaranteed that Q is reachable. So, we only need to calculate the Us. By reading its definition above, we note that $U_{t,i}$ is equal to the longest path from P to t using teleporters as intermediate steps, where the length of each step is simply the distance between the two points.

We can calculate the length of the required longest paths for all t and a fixed i in $O(\mathbf{N}^3 \log i)$ time by using something similar to iterated squaring to calculate the matrix of largest distances from any teleporter to any other in i - 1 steps, and then combining that with the vector of distances from P to each teleporter. The "multiplication" here is not an actual matrix times matrix multiplication, but rather the use of the property that the longest path from t to u in i steps is the longest path from t to v in j steps plus the longest path from v to u in k - j steps, for some v. Taking j = k / 2 for even k shows how to do log k steps overall. This, combined with a binary search on the number of steps, gives an algorithm with overall time complexity $O(\mathbf{N}^3 \log^2 M)$. If you have a good implementation in a fast language, this runs in minutes, but it's enough to pass the Large.

It's possible to get rid of one the log factors for an overall time complexity of $O(\mathbf{N}^3 \log M)$, and a program that finishes the Large in a few seconds. This is achieved by starting the binary search on a range [min, max) whose size, that is, max - min, is a power of 2. Each step calculates mid as the averge of min and max, so mid - min and max - mid are also powers of 2, which proves by induction that the range size is a power of 2 in every step of the binary search. Then, since mid - min is also a power of 2 in every step, every distance matrix you need is of a number of steps that is itself a power of 2 (the range keeps being cut in half, so it remains of size a power of 2, so the delta between the min and the midpoint that we need to test is always a power of 2). That means we can precalculate all needed matrices in $O(\mathbf{N}^3 \log M)$ time, since the matrix for 2^{k+1} steps is the "square" of the matrix for 2^k steps. With the memoized matrices, each step of the binary search only takes $O(\mathbf{N}^2)$ time to "multiply" the matrix and the initial vector.