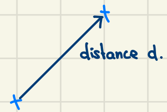
- move the ball to a target location t

using as few shots as possible.

Each shot you can move the ball through the air by a distance of at most d.



First solution idea:

Construct graph of the n safe locations on the playing grounds:

iff $\text{dist}(p_i,p_j)^2 \leq d^2$.

$O(n^2)$ construction

Run BFS from target location in $O(n+m) = O(n^2)$

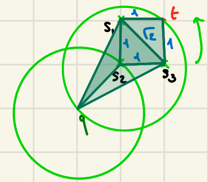
For each of the m starting locations: $O(m)$

For each of the n safe locations on the playing ground in $O(n)$

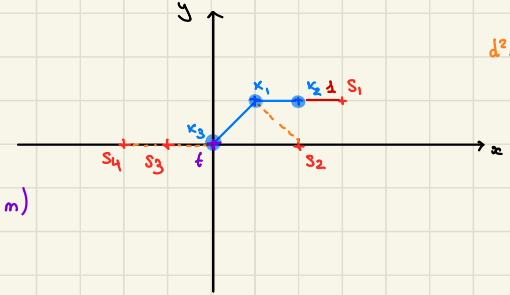
check whether $\text{dist}[s_i] \leq k-1$.

- For the first group of test cases we have $k=2$:

Construct a Delaunay triangulation of safe locations on playing grounds $\{s_1, \dots, s_n\}$ in $O(n \log n)$.



$O((n+m)\log(n+m) + n)$.



$d^2 = 2 = q.$

- for $k=2$ we can use at most two shots:

{ 1 shot from starting position to safe point.
  1 shot from safe point to target

OR

{ 1 shot from starting position to target.

Set of safe points reachable from target

for $k=1$: the question becomes whether the start position is within distance d to the starting point.

for $k=2$: the question becomes an intersection:

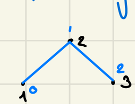
$\{$all points reachable from the start$\}$

$\cap$ $\{$all points reachable from the target$\} \neq \emptyset$.

$$\underset{(\text{start})}{s} \xrightarrow{\text{dist} \leq d} \underset{(\text{some point})}{x} \xrightarrow{\text{dist} \leq d} \underset{(\text{target})}{t}$$

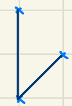
dist $\leq d$.

This works for the first group of test cases ✓.



For the second group of test cases we assume $k = n$.

- That means we can go through a Spanning Tree of a graph.
  We know that the EMST $\subseteq$ Delaunay Triangulation.
- Hence we only care whether t and s_i are in the same connected component of the unit disk full distance graph.



- We can compute the connected component C_t of the full unit disk graph where $dist(p_i, p_j)^2 \leq q = d^2$.
- Then we construct a Delaunay Triangulation on C_t to check whether q is within a distance of q away from this set of points.

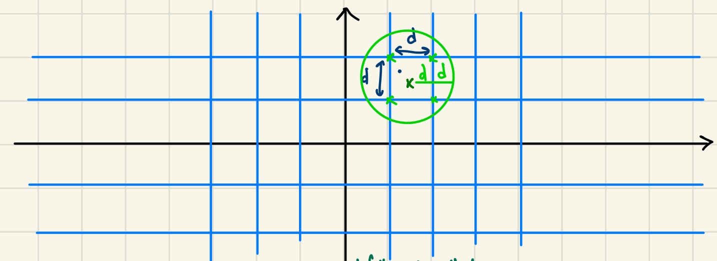
The question becomes: is s within d^2 away from C_t?

Runs in $O(n\log n + n + n\log n + m\log n)$
$= O(n\log n + m\log n)$

For the third and fourth group of testcases:

- Delaunay triangulation does not capture distance hops.
  For this we need to try something else: running BFS on the partitioned $\mathbb{R}^2$ plane.

$\left\lceil \frac{x_{max}}{d} \right\rceil \cdot \left\lceil \frac{y_{max}}{d} \right\rceil$ buckets which represent squares in the plane.



it follows here that
the nearest points must be
in the eight neighbouring squares
OR the square itself.

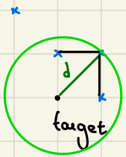
The coordinates are $\left\lfloor \frac{x}{d} \right\rfloor , \left\lfloor \frac{y}{d} \right\rfloor$ for the corresponding bucket.

- Run BFS on this graph by checking whether the neighbour point is a distance of at most d from the current point and transition to next point.

$\Rightarrow$ This identifies all the points that are $k-1$ hops away from t.

Ideally this runs in $O(n)$. Then we construct the Delaunay triangulation of these $k-1$ hop points in $O(n\log n)$. for the m point queries this will be $O(m\log n)$

• We note that for small enough k values $k \leq 2$, $k \leq 20$ we still get some points.
this hints that k might be used as a parameter in the total runtime complexity of the algorithm.



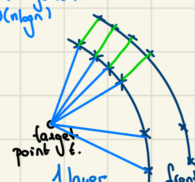
$O(n) \leq d$ of t.

→ compute DT $O(n\log n)$
and query for nearest vertex
to that DT in $O(n\log n)$

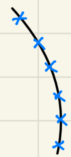
runs in total in $O(n\log n)$

→ repeat k times and
this would give you the
$k-1$ hop neighbours.

Construct the Delaunay triangulation $O(n\log n)$



up to $k-1$ layers.



Call
v.nearest_vertex()
in $O(n\log n)$
and check whether
nearest vertex is within
distance $\text{dist}(v, q)^2 \leq d^2$.

This process runs in $O(k \cdot n \cdot \log n)$.

$\log(4) + 4\log(10)$
$\leq 2 + 4 \cdot 4 = 18$.

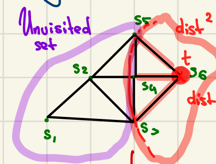
$20 \cdot 4 \cdot 10^4 \cdot \log(4 \cdot 10^4)$
$= 80 \cdot 10^4 \cdot 18 = 10^5 \cdot 8 \cdot 80 = 160 \cdot 10^5 = 10^7$.

(•) For the third and fourth group of testcases:

Our time complexity $O(kn\log n)$ is too bad and time limits.
⤳ Clearly the inefficiency in this approach is that we are reconstructing the Delaunay triangulation in $O(n\log n)$ at every step along the way. ⇒ $O(kn\log n)$.
⤳ Can we somehow achieve a BFS traversal only constructing the Delaunay Triangulation of the safe points $\{s_1, \ldots, s_n\}$ once in $O(n\log n)$?
⤳ Can we then query for the starting points $\{a_1, \ldots, a_m\}$ in $O(m\log n)$?

Transform $DT\{s_1, \ldots, s_n\}$ into $DT\{s_i : s_i$ is reachable from t in $k-1$ hops$\}$.

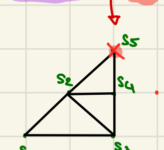
DT is a proximity search structure ⇒ good at answering how far am I to a set of points?



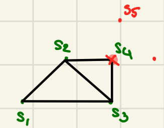
$q = \{t\}$

Visited Set

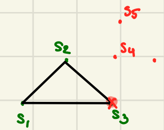
• Question: for the vertices in the unvisited set we want to know how close they are to the visited set?



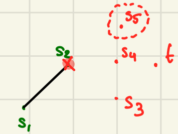
$q = \{t, s_5\}$



$q = \{t, s_5, s_4\}$



$q = \{\cancel{t}, s_5, s_4, s_3\}$



$q = \{s_5, s_4, s_3\}$

If we have t.nearest_vertex(p) > d
then the unvisited set is too far away
⇒ we can stop querying.

We stop visiting if dist[u] = k-1.

Clearly this is a general algorithm to find points k-1 hops away.
⇝ BFS

```
dist[t] = 0
dt.remove(t)      O(logn + deg(t)) = O(logn)
q.push(t)
 while(!q.emty()) {
     int u = q.front();        O(1)
     q.pop();
     if(dist[u] >= k-1) continue;
     while(!dt.empty()) {
         int v = dt.nearest_point(u);   O(logn)
```

```
if( dist(u,v) ≤ d ) {
    dist[v] = dist[u] + 1
    dt.remove(v)            O( logn + deg(v) )
    q.push(v)
    reachable.push_back(v)
}
}
```

$$O\Big(\sum_{v} \big(1 + \log n + \deg(v) \big) + n\log n \Big)$$
$$= O(n + n\log n + n + n\log n)$$
$$= O(n\log n).$$

⇝ notice that DT construction is $O(n\log n)$.

⇝ once we get k-1 hop reachable set we can construct DT of it $O(n\log n)$.

⇝ Query m times that DT in $O(m\log n)$.

$$O(n\log n + n\log n + n\log n + m\log n)$$
$$= O(n\log n + m\log n).$$