1. *Algorithm Description*

We proceed by modeling the network of trails as a directed graph $G(V,E)$ where each edge $(u,v) \in E$ has an associated value $\mathbb{E}(u,v)$ that represents the expected number of slide occurrences while traversing (u,v) and visiting v. By modeling this way, we remove the weights on the vertices (and aggregate them on the edges) in such way that an edge (u,v) with terminal incidence at v has an assigned value of:

$$\mathbb{E}(u,v) = w(u,v) + w(v) \tag{1}$$

With this configuration in mind, we aim to find a path p such that the expectation on the number of slide occurrences on p is minimized. Let s be the source and t be the terminal, and let p be the sequence $(v_0, v_1, \cdots, v_k)$, where $v_0 = s$ and $v_k = t$, then:

$$p = \arg\min \left(w(s) + \sum_{i=1}^{k} \mathbb{E}(v_{i-1}, v_i) \right) \tag{2}$$

We use Dijkstra's algorithm to solve the single source shortest path problem on the converted graph. Using Equation 2 the following setup is assumed:

- *Vertices* — unchanged sites on the map with no associated probabilities nor weights except for s which has an assigned weight $w(s)$ representing the expected number of slides in s.
- *Edges* — directed routes between different sites of the map with edge weights given by (1).

We set the source s to be the marked position on our friends on the map and initialize the $cost(s)$ to $w(s)$ as we must account for the expectation in s. Then we exhaust all paths going from s to the remaining vertices of G and (since edge weights encode expectations) we pick the smallest path with terminal at a trailhead.

Time Analysis

Let V represent the different sites on the map and E the routes between them. Then we can construct G in $\mathcal{O}(V+E)$ time. A call to Dijkstra on G takes $\mathcal{O}((E+V)\log V)$ and the reconstruction of p takes at most $\mathcal{O}(V)$ with backtracking. This yields a total running time of $\mathcal{O}((E+V)\log V)$.

2. Consider the following definition:

> **Definition —** Let $G = (V,E)$ be an undirected graph with edge-weights given by $w\colon E \to \mathbb{R}$. Assume that $w(e) \neq w(f)$ whenever e, f are distinct edges of G. We say that an edge is *treacherous* if it is the maximum weight edge for some cycle of G. On the other hand, an edge is *reliable* if it is not contained in any cycle of G.

> **Claim 1 —** The minimum spanning tree of G contains every *reliable* edge.

Proof. Assuming G is connected, we prove the more general claim P that "an edge e is *reliable* in G if and only if it belongs to every spanning tree of G." Note Claim 1 follows as a direct consequence of P.

$\Longrightarrow$ (*Sufficient Condition*) — Suppose e does not belong to every spanning tree of G. Let T be a spanning tree that does not contain e. Then the spanning tree T is a subgraph of $G \setminus \{e\}$. It follows then that for any arbitrary vertices u, v in $G \setminus \{e\}$ there is a unique (u,v) path in T. This is also a (u,v) path in $G \setminus \{e\}$ since $T \subseteq G \setminus \{e\}$. Thus, $G \setminus \{e\}$ is connected, and e is not *reliable* in G.

$\Longleftarrow$ (*Necessary Condition*) — If e is not *reliable* (and part of cycle) in G then its removal causes $G \setminus \{e\}$ to remain connected. So $G \setminus \{e\}$ must have a spanning tree. But such a spanning tree would also be a spanning tree of G since it has the same vertex set, but it wouldn't contain e.

□

Claim 2 — The minimum spanning tree of G does not contain any *treacherous* edge.

Proof. Suppose (for the sake of contradiction) that there is a minimum spanning tree, say T, containing a *treacherous* edge $e = (u, v)$ of some cycle C of G.

Let $T' = T \setminus \{e\}$. Then T' has two connected components. Because e was part of cycle C, there is a path connecting u and v in G not containing e. This path must contain an edge with endpoints in different connected components of T'. Let such edge be e'. Because e' and e are in C and by assumption e is the edge with maximum weight in C, it follows that $w(e') < w(e)$.

Construct $T'' = T' \cup \{e'\}$. Note T'' is connected as e' created a path between the two connected components of T'. Additionally, T'' has the exactly number of edges as T, so T'' is a valid spanning tree of G. But its weight:

$$w(T'') = w(T) - w(e) + w(e') < w(T)$$

Thus T'' is a spanning tree of G with smaller weight that T—the minimum spanning tree of G. But this contradicts the initial supposition of minimality of T in G. □

Algorithm Description

We proceed as follows. We iteratively delete the highest *treacherous* edge until the resulting graph becomes acyclic. Since we only delete *treacherous* edges, the resulting graph stays connected. Since we stop when the graph becomes acyclic, the output will indeed be a spanning tree. Since the only edges removed were edges by Claim 2 not contained in any minimum weight spanning tree, the resulting tree must have minimum weight and be the MST of G. The pseudocode for this is provided below.

Algorithm Pseudocode

Algorithm 1 Minimum-Spanning-Tree($G \leftarrow (V, E)$)

```
T ← Reverse-Sort-By-Weight(E)
for e ∈ T do
    if T ∖ {e} is connected then
        T ← T ∖ {e}
    end if
end for
return T
```

Time Analysis

The implementation of Algorithm 1 begins by sorting the edges in $\mathcal{O}(E \log E)$ time. For each edge e we check whether $T \setminus \{e\}$ is connected and we do so via DFS. Each call to DFS takes $\mathcal{O}(V + E)$, and doing for all entries of E takes $\mathcal{O}(E(V + E))$. Since G is connected, E dominates V asymptotically, so the total running time is $\mathcal{O}(E^2)$.