A Complete Lean Formalization of a TCS Proving Benchmark

José Luis Delgado [1]

Abstract

We formalize the public Phase 1 benchmark for Track 2 of the ICML 2026 AI for Math Workshop, Theoretical Computer Science Proving in Lean. The benchmark consists of 29 challenge files, worth 500 points, covering treaps, segment trees, binary heaps, Dijkstra's shortest-path algorithm, and Kruskal's minimum-spanning-tree algorithm. The artifact checks in the supplied Lean 4.28.0, CSLib, and mathlib environment, with no remaining `sorry` or `admit` in the challenge files and no changes to the provided definition files. The development centers on abstraction relations and inductive invariants that connect executable data structures to their mathematical specifications: heap membership and priority invariants for Dijkstra, and a disjoint-set/forest connectivity invariant together with a cut-property exchange argument for Kruskal. We describe the proof organization and the library interfaces suggested by the development.

1. Introduction

The public Track 2 benchmark of the ICML 2026 AI for Math Workshop consists of pre-formalized Lean statements in theoretical computer science (AI4Math @ ICML 2026, 2026). The tasks are organized around executable algorithms and data structures: randomized search trees (Seidel & Aragon, 1996), segment trees, binary heaps, Dijkstra's algorithm (Dijkstra, 1959), and Kruskal's algorithm (Kruskal, 1956). These problems match a recurring bottleneck in verified algorithms in Lean (Moura & Ullrich, 2021; The mathlib Community, 2020): proofs must relate concrete implementations to abstract invariants over sets, graphs, paths, and weights before the final mathematical statements can be closed.

We give a complete solution of the public Phase 1 benchmark, built on CSLib (Barrett et al., 2026). The submitted artifact contains exactly the 29 required challenge files, compiles in the supplied repository, and leaves all `Def_*` files untouched. The proof development records the mathematical infrastructure needed for the benchmark: abstraction relations for executable data structures, preservation lemmas for recursive operations, and graph-theoretic invariants for the final algorithmic correctness statements. Among the problem families, Dijkstra and Kruskal require the largest proof infrastructure: their public statements are compact, yet their proofs require intermediate invariants that connect the executable representation with the mathematical object being specified.

Contributions. The artifact discharges the public Phase 1 benchmark: 29 challenge files, 500 total points, checked by Lean under the permitted axioms. The development uses a recurring pattern for algorithmic verification in Lean: define an abstraction relation for the executable representation, strengthen the recursive invariant until it matches the standard mathematical proof, and derive the public correctness theorem as a specialization. The two largest proofs instantiate this pattern in different ways: Dijkstra uses a queue invariant with a boundary condition for settled vertices, and Kruskal uses a disjoint-set tracking invariant with a safe-forest exchange theorem.

2. Benchmark

The benchmark consists of five problem families, summarized in Table 1. Each challenge file originally contains one scoring obligation, while definition files provide the fixed environment. Scoring is binary per challenge: a file receives its points exactly when the proof compiles under the allowed axiom set.

The lower-level challenges put the executable definitions into proof-ready form: heapify preserves heap order, `extract_min` returns a valid heap, segment tree queries agree with interval coverage, and treap split/merge operations preserve the binary-search-tree and heap conditions. The high-value correctness theorems use this local infrastructure, with additional global invariants for the final Dijkstra and Kruskal statements.

[1]Universitat Oberta de Catalunya. Correspondence to: José Luis Delgado <jdelgado13@uoc.edu>.

The 3rd *AI for Math Workshop* at the 43rd International Conference on Machine Learning (ICML), Seoul, South Korea, 2026.
Copyright 2026 by the author(s).

Table 1. Public Phase 1 benchmark.

Family	Challenges	Points
Treap algorithms and analysis	17	100
Segment tree	4	100
Binary heap	4	100
Dijkstra	3	100
Kruskal MST	1	100
Total	29	500

Table 2. Proof obligations beyond local simplification.

Target	Main additional infrastructure
Dijkstra	Heap membership; no-duplicate preservation; priority-update heap preservation; boundary invariant; shortest-path witness lemma.
Kruskal	Union-find semantic tracking; selected-forest acyclicity; cut lightness from sorted scan; spanning-tree exchange; weight-sum comparison.
Treap analysis	Probability decomposition over ancestor events; expected-depth summation.
Segment tree query	Coverage-interval semantics; agreement between recursive query structure and interval membership.

Executable recursion and graph proofs. Many smaller obligations close by structural induction aligned with the executable definition. The final graph-algorithm theorems have induction principles generated by recursive programs and correctness arguments drawn from textbook graph proofs. The formalization supplies the interface between these two levels. Table 2 lists the main proof obligations that had to be made explicit before the final statements became tractable.

3. Formalization Architecture

The formalization follows a standard refinement pattern: executable definitions are related to mathematical specifications by abstraction relations, and recursive programs are verified using inductive invariants stated over the abstract view.

Abstraction relations. For binary heaps and union-find forests, the executable representation differs from the object used in the correctness statement. Proofs about graph reachability or shortest paths use explicit bridge predicates. For Dijkstra, heap membership is represented through the underlying binary tree, together with no-duplicate and heap-order invariants. For Kruskal, the union-find state is related to reachability in the current forest.

Strengthened invariants. Direct induction on the executable recursion yields hypotheses that are too weak for the final theorem statements. The proofs use stronger invariants preserved by each recursive step; the public theorems are obtained by instantiating these invariants at the initial state.

Mathematical exchange principles. The largest graph-algorithm proofs require mathematical lemmas outside local simplification. Dijkstra extracts a vertex whose tentative distance is already exact. Kruskal accepts an edge that can be exchanged into a minimum spanning tree. These statements carry the graph-theoretic content of the algorithms.

4. Case Study: Dijkstra

The Dijkstra challenge states correctness of the final distance map produced by the executable implementation. The proof uses a strengthened invariant over a distance function `dist` and a binary heap queue `q`. The invariant states:

1. The queue is a min-heap ordered by `dist`.
2. Queue contents are duplicate-free.
3. The source has distance zero.
4. All tentative distances are upper bounds on true graph distance.
5. Every settled vertex already has exact distance.
6. Every edge from the settled region to the queue has been relaxed.

The final clause is the boundary invariant, recording the fact preserved by relaxing all neighbors of the extracted vertex in terms of the queue itself.

In mathematical notation, the invariant has the following shape:

$$\begin{aligned}&\mathsf{Heap}(q, \mathsf{dist}) \wedge \mathsf{NoDup}(q) \wedge \mathsf{dist}(s) = 0 \wedge \\ &(\forall x.\ \delta(s, x) \leq \mathsf{dist}(x)) \wedge \\ &(\forall x.\ x \notin q \Rightarrow \mathsf{dist}(x) = \delta(s, x)) \wedge \\ &(\forall p, z.\ p \notin q \wedge z \in q \wedge p \sim z \Rightarrow \\ &\qquad \mathsf{dist}(z) \leq \delta(s, p) + 1).\end{aligned}$$

The final conjunct is the boundary condition, the formal version of the textbook assertion that all outgoing edges from the settled set have been considered.

Heap bridge. The heap library proves local preservation properties for heap operations. Dijkstra also needs semantic information about the queue as a finite set of vertices. The proof adds a list model for binary trees, membership equivalences, no-duplicate preservation, and exact extraction facts. These lemmas imply that extraction removes a vertex from

the queue, that the remaining heap members were already present, and that the root of a min-heap is an argmin among queued vertices.

Priority updates require a separate heap argument. Relaxation changes `dist`; a heap ordered by the old priority function need not be ordered by the new one. Congruence and update lemmas show that priority changes preserve heap order when all non-updated vertices keep their priority and the updated priority only decreases.

Exactness of extraction. The central graph lemma is a shortest-path witness argument. For any queued vertex `u`, take a shortest path from the source to `u`. If the source is queued, it is the witness. Otherwise, the first queued vertex on the path has a settled predecessor, and the boundary invariant gives an upper bound on its distance. Together with the heap argmin property and the global upper-bound invariant, this proves that the extracted vertex has exact distance.

Recursive correctness. After extraction, relaxing the outgoing neighbors preserves the source-zero, upper-bound, settled-exact, boundary, heap, and no-duplicate invariants. A strong induction on the queue size proves correctness of the recursive implementation. The public theorem follows from the initial queue containing all vertices and the initial distance map that is zero at the source and infinity elsewhere.

5. Case Study: Kruskal

The Kruskal challenge is a single 100-point theorem: the graph produced by the executable algorithm is a minimum spanning tree. The proof has three layers: disjoint-set semantics, forest acyclicity/connectivity, and optimality by the cut property.

Disjoint-set abstraction. The implementation stores components in a union-find forest, while MST correctness is stated in terms of graph connectivity. The proof uses an abstraction relation saying that two vertices have the same union-find representative exactly when they are reachable in the current selected-edge forest:

$$\mathsf{find}_{UF}(x) = \mathsf{find}_{UF}(y) \iff \mathsf{Reach}_F(x, y).$$

The precise effect of `union` is needed because after joining the two endpoint components, the new equivalence relation is exactly the old one plus the cross-component identifications. The abstraction relation is preserved across the scan of the sorted edge list.

Forest invariant. Whenever Kruskal accepts an edge, its endpoints are in distinct union-find components. By the tracking invariant, they are not reachable in the current forest, so adding the edge preserves acyclicity. At the end of the scan, every original graph edge has endpoints in the same final union-find component; connectedness of the input graph lifts this local fact to connectedness of the selected forest. Acyclicity and connectedness give the spanning-tree property.

Cut property and exchange. Minimality uses a safe-forest invariant: after processing a prefix of the sorted edge list, for every spanning tree of the original graph there exists a spanning tree of no greater weight containing the current selected forest:

$$\begin{aligned}\forall T.\ &\mathsf{SpanningTree}(T, G) \Rightarrow \exists T'.\\ &\mathsf{SpanningTree}(T', G) \wedge F \subseteq E(T') \wedge\\ &w(T') \leq w(T).\end{aligned}$$

When Kruskal rejects an edge, the invariant is unchanged. When it accepts an edge, the accepted edge is safe: it is light across the cut induced by the component of one endpoint in the current forest. The sorted scan gives lightness, since any earlier crossing edge would already have connected the components, contradicting acceptance.

The exchange lemma carries the optimality argument. Given a spanning tree containing the current forest, add the accepted light edge; the resulting cycle contains a crossing tree edge. Removing that crossing edge gives another spanning tree that contains the enlarged forest and has no larger weight. In Lean this requires explicit lemmas about edge subsets, `Sym2` endpoint orientation, deleting an edge from a tree path, reconstructing a spanning tree after the swap, and comparing weight sums over equal edge sets. Induction over the sorted edge list then closes the safe-prefix invariant, and the final tree is minimum.

6. Results

The artifact verifies the entire public Phase 1 benchmark. The verification conditions are:

- All 29 `Challenge_*.lean` files compile.
- No `sorry` or `admit` occurs in challenge files.
- The provided `Def_*` files are not modified.
- The repository builds under Lean 4.28.0 with the supplied CSLib/mathlib environment.
- The proof uses only the permitted axioms: `propext`, `Quot.sound`, and `Classical.choice`.

Table 3. Verification status.

Family	Solved	Score
Treap algorithms and analysis	17/17	100/100
Segment tree	4/4	100/100
Binary heap	4/4	100/100
Dijkstra	3/3	100/100
Kruskal MST	1/1	100/100
Total	29/29	500/500

7. Conclusion

We give a complete Lean proof-engineering solution for the public Phase 1 TCS proving benchmark. The final artifact closes all 29 challenge files, worth 500 points, under the supplied environment and permitted axioms. The main technical content is the conversion of executable data-structure code into semantic invariants strong enough to support textbook graph-algorithm arguments. The Dijkstra and Kruskal developments are small libraries of representation bridges, preservation lemmas, and exchange principles, from which the public theorem statements follow.

References

AI4Math @ ICML 2026. 3rd AI for math workshop: Toward self-evolving scientific agents. https://ai4math2026.github.io/, 2026. ICML 2026 workshop, Seoul, South Korea, July 11, 2026; accessed 2026-05-25.

Barrett, C., Chaudhuri, S., Montesi, F., Grundy, J., Kohli, P., de Moura, L., Rademaker, A., and Yingchareonthawornchai, S. CSLib: The Lean computer science library, 2026. URL https://arxiv.org/abs/2602.04846.

Dijkstra, E. W. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959. doi: 10.1007/BF01386390. URL https://doi.org/10.1007/BF01386390.

Kruskal, Joseph B., J. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):48–50, 1956. doi: 10.1090/S0002-9939-1956-0078686-7. URL https://doi.org/10.1090/S0002-9939-1956-0078686-7.

Moura, L. d. and Ullrich, S. The Lean 4 theorem prover and programming language. In Platzer, A. and Sutcliffe, G. (eds.), *Automated Deduction – CADE 28*, volume 12699 of *Lecture Notes in Computer Science*, pp. 625–635, Cham, 2021. Springer International Publishing. ISBN 978-3-030-79876-5. doi: 10.1007/978-3-030-79876-5_37. URL https://doi.org/10.1007/978-3-030-79876-5_37.

Seidel, R. and Aragon, C. R. Randomized search trees. *Algorithmica*, 16:464–497, 1996. doi: 10.1007/BF01940876. URL https://doi.org/10.1007/BF01940876.

The mathlib Community. The Lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2020, pp. 367–381, New York, NY, USA, January 2020. Association for Computing Machinery. ISBN 9781450370974. doi: 10.1145/3372885.3373824. URL https://doi.org/10.1145/3372885.3373824.

A. Reproducibility Checklist

The submitted code artifact is a zip file containing the 29 challenge files matching `Challenge_*.lean`. To reproduce the check, unpack the artifact into the official challenge repository so that the supplied challenge files replace the corresponding files under `Challenges/`. The full repository build is then checked with:

```
lake build
```

The absence of placeholders in challenge files is checked with:

```
grep -RInE 'sorry|admit' Challenges
    --include 'Challenge_*.lean'
```

These commands run against the challenge repository configured for Lean 4.28.0, CSLib, and the corresponding mathlib version.