Building a maze

We want to build a maze out of a rectangular grid by removing some walls.

There should be exactly one path from start to goal cell.

Algorithm:

- Consider all walls in random order.
- If the two cells separated by the wall are not yet connected, then remove the wall.
- May stop when start and goal have become connected.

To implement this algorithm efficiently, we maintain the subsets of cells that are connected.

We need two operations:

- Determine whether two cells are in the same subset,
- Replace two subsets by their union

Union-Find data structure

Given a "universe" U of n elements, we want to maintain a partitioning of U into disjoint subsets.
At the beginning, each element is in its own subset.
We support two operations:

- find(x): determine which subset contains x
- union(s, t): replace subsets s and t by their union

Applications:

- Building a maze
- Minimum spanning tree
- Nearest Common Ancestor

Quick-find

Quick-find data structure:
Create an array `A` with n slots. The value `A[i]` is the subset containing element i.

Find takes constant time.

Union takes $O(n)$ time.

Quick-union

Quick-union data structure:
We organize each subset as a tree.

Union takes constant time.

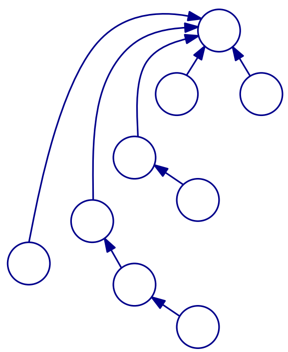
Find needs to trace references to the root in time $O(h)$, where h is the height of the tree.

Union by size heuristic: When performing a union, make the smaller tree a subtree of the root of the larger tree.

This heuristic guarantees that a tree of size m has height $O(\log m)$.

Path compression

Path compression heuristic:
During a find operation, we make all the nodes found children of the root.



It is surprising that find should modify the tree. The idea is that this will improve the running time of future find operations.

Time complexity

Let $F(m, n)$ be the total number of parent links followed by m find operations (in a sequence of union and find operations) on a universe of size n.

Theorem: If $n < 2^{2^{2^{2^{2}}}}$, then $F(m, n) \leq 4m + 4n$.

But $F(n, n)$ is not a linear function, and for $n \to \infty$ we have $F(n, n)/n \to \infty$.

The true time complexity is $F(m, n) = O(n + m\alpha(m + n))$, where $\alpha(m)$ is the inverse of Ackermann's function. It grows very very slowly.

Analysis of path compression

When we create a link from u to v during a union, we assign rank $r(u) = \lfloor \log n(u) \rfloor$ to u, where $n(u)$ is the number of nodes in the tree with root u.

Lemma: If v is the parent of u at some time and v is not a root, then $r(v) > r(u)$.

Lemma: The number of nodes with rank $r(u) = s$ is at most $n/2^s$.

Proof: When the rank is assigned, u is the root of a subtree with at least 2^s nodes.

After the union, these nodes are part of a tree with at least 2^{s+1} nodes, and can never be counted again when assigning rank s.

Counting acorns

When we assign the rank, we also give some acorns to u:

Group	Ranks	Acorns	Total acorns
0	0	0	0
1	1...4	$4 - r(u)$	$\leq 2.125n$
2	5...16	12	$\leq 0.75n$
3	17...65536	65536	$\leq n$

The total number of acorns given to nodes is $\leq 4n$.

A find operation follows at most three links between different groups, and one link to the root of the subtree $\Rightarrow 4m$ links.

Following a link from u to v in the same group is paid with an acorn at u.

$F(m, n) \leq 4m + 4n$.