occurrence once.

represented as linked lists. Very expensive to remove the dupes and make sure we only have each element

When in doubt, use a linked list, but that's inherently inefficient. Consider doing set union of $S$ and $T$

In the absence of an ordering property, how do we handle data efficiently?

two children, it's also the global max of the entire heap).

example, the root of a max-heap is not just the largest element of the three children subtree of itself and its
determined in sorted order, and/or to imply a transitivity from a local property to a global property (for
elements in sorted order, and/or to imply a transitivity from a local property to a global property (for

most algorithms rely on some sort of “less than” condition that allows one to bound a search, to keep

deal with the unordered-ness of elements in sets

• take the union of two sets $S$ and $T$

• determine whether an element $x$ is a member of a set $S$

A classic example would be a collection of sets. When working with sets, we need to be able to

There are many instances in which we have a set, or a bag, or some similar object that must be manipulated.
will be our "handle" on the class, analogous to the root of a max-heap.

Inevitably, what we will want to have will be a canonical representation for each equivalence class. This

get run over, so the network is dynamic.

may be almost the same thing, the Chicago freeways system at rush hour.) Sensors break, lose battery charge,

Consider the last of these as a good dynamic example: lots of sensors dropped on a battleheld (or, what

Membership in a cycle/digraph/connected component in a graph

Congruence classes of the integers modulo an integer m

Membership in a set

Equality of elements

Examples:

transitive: &ab and &be &ae reflexive: &aa is true

symmetric: &ab &ba

Definition 11.1. An equivalence relation is a binary relation R on a set S that is
\textbf{Find}(x) \quad \text{to find the representative for the set of which } x \text{ is a member.}

\textbf{Union}(S, T) \quad \text{to create the union of sets } S \text{ and } T \text{ and return a representative for the union.}

\textbf{Make-Set}(x) \quad \text{to create a new set } S \text{ in the collection, with the only member of } S \text{ being a new element } x.

We need at least three operations:

\begin{itemize}
  \item \textbf{Find}(x)
  \item \textbf{Union}(S, T)
  \item \textbf{Make-Set}(x)
\end{itemize}

As chosen, as long as we can deal with it in a predictable way.

A \textit{disjoint-set data structure} maintains a collection \( S \), \( T \), \ldots, \( S_1 \) of disjoint dynamic sets. Each set will be identified and manipulated through a \textit{representative}. It doesn't really matter how the representative
that is created, as the next examples show.

One thing that is a big deal is that the representation of the graph can have a major effect on the structure.

are supposed to be equivalent, then we do a \texttt{Union} operation to join the two trees together into one.

If we issue a \texttt{Find} for two nodes \( b \) and \( q \), and we choose parents to different roots, and yet these two nodes

The \texttt{Find} operation chooses parents from a node until it finds the root.

We start with a Graph \( G \) of nodes \( V \) and edges \( E \). Each node starts as the root of its own tree.

testing the equivalence relation is "fast".

The \texttt{Union-Find} algorithm is an algorithm for constructing a data structure with online data for which
we issue "edges" $AB, BC, CD, DE, EF, FG, GH, JK, KL$ in order, then we get the structures:

Let's assume that we have sets $\{A, B, C, D\} \subseteq \{A, B, C, D, E, F, G, H\}$ and $\{J, K, L\} \subseteq \{A, B, C, D, E, F, G, H\}$. Then, we can represent in a data structure.

11.1. Example: Set Membership
If instead we presented "edges" AB, CD, EF, GH, AC, EG, AE, JK, IL, then we might get

This "works" but is clearly inefficient. We choose more pointers than is perhaps necessary to get from C to A.

\[ H \]
\[ \downarrow \]
\[ G \]
\[ \downarrow \]
\[ F \]
\[ \downarrow \]
\[ E \]
\[ \downarrow \]
\[ D \]
\[ \downarrow \]
\[ C \]
\[ \downarrow \]
\[ B \]
\[ \downarrow \]
\[ A \]

and eventually
structure would be best:

this was a set operation so that membership in the tree was the highest priority issue, then probably a halt
which would have better depth searching (and characteristics. And if searching was a major priority, and

H

| D |
|   |
| C |

| B |
|   |
| C |

| T |
|   |
| K |

|     |
|     |
|     |

connected component of the graph will be as follows.

If we present the edges as above and do no path compression, then the final union-find tree for this

\[ A \rightarrow B \rightarrow C \rightarrow D \rightarrow B \]

and the \( B \rightarrow A \) will be cancelled. 

Learning the cycle \( BCDH \) as \( B \rightarrow C \rightarrow D \rightarrow H \rightarrow B \). Then the

orientation, perhaps with \( BCD \) and \( BDH \) as \( B \rightarrow D \rightarrow H \rightarrow B \). By using a union-find algorithm, (for sums of cycles, consider the following: trace out the cycles with an

We can generate a spanning set of cycles such that the sums of cycles comprise all the cycles

This graph has cycles with nodes \( BCD, BCD, CDE, DCE, DE, EDC \).

\{ \{B, C, D\}, \{A, B, C, D, E\}, \{C, F, G\} \}

Now consider a graph with nodes

11.2 Example: Cycles in Graphs
and so six cycles in all.

This gives a set of three cycles \( C_1, C_2, C_3 \), there are three sums of these cycles \( C_1 \oplus C_2, C_1 \oplus C_3, C_2 \oplus C_3 \), try to add the edge \( H \) we will get a cycle.

Try to add the edge, we will get a cycle. Similarly, when we try to add the edge \( D \) we will get a cycle, and when we add the cycle and discard the edge. Instead of adding the edge, we write out "find" we will return the same root for the node and the node. Instead of adding the edge, we can tell this because the cycle \( BCD \) we add edges, when we try to add \( CD \), this would produce the cycle \( E \)
Path Compression

technique for improving the "find" process.

Trees compared with the time to search the forest of trees. And many variations exist on pointer jumping of trees compared with the time to search the forest of trees. As well as the time to build the forest of trees. It is relevant to consider the number of nodes and number of edges, as well as the structure of the graph to be preserved. It