Disjoint sets

- A set is a collection of elements (no-repeats)
- In computer science, two sets are said to be disjoint if they have no element in common.
  - \( S_1 \cap S_2 = \emptyset \)
- For example, \( \{1, 2, 3\} \) and \( \{4, 5, 6\} \) are disjoint sets.
- For example, \( \{x, y, z\} \) and \( \{t, u, x\} \) are not disjoint.

Partitions

- A partition \( P \) of a set \( S \) is a set of sets \( \{S_1, S_2, \ldots, S_n\} \) such that every element of \( S \) is in exactly one \( S_i \).
  - Put another way: \( S_1 \cup S_2 \cup \ldots \cup S_n = S \)
  - \( i \neq j \) implies \( S_i \cap S_j = \emptyset \) (sets are pairwise disjoint)
- Example:
  - Let \( S \) be \( \{a, b, c, d, e\} \)
  - One partition: \( \{a\}, \{d,e\}, \{b,c\} \)
  - Another partition: \( \{a,b,c\}, \emptyset, \{d\}, \{e\} \)
  - A third: \( \{a,b,c,d,e\} \)
  - Not a partition: \( \{a,b,d\}, \{c,d,e\} \) … element \( d \) appears twice
  - Not a partition of \( S \): \( \{a,b\}, \{c,e\} \) … missing element \( d \)

Binary relations

- \( S \times S \) is the set of all pairs of elements of \( S \) (cartesian product)
  - Example: If \( S = \{a,b,c\} \)
    - then \( S \times S = \{(a,a),(a,b),(a,c),(b,a),(b,b),(b,c),(c,a),(c,b),(c,c)\} \)
- A binary relation \( R \) on a set \( S \) is any subset of \( S \times S \)
  - i.e., a collection of ordered pairs of elements of \( S \).
  - Write \( R(x,y) \) to mean \( (x,y) \) is in the relation.
  - (Unary, ternary, quaternary, … relations defined similarly)
- Examples for \( S = \text{people-in-this-room} \)
  - Sitting-next-to-each-other relation
  - First-sitting-right-of-second relation
  - Went-to-same-high-school relation
  - First-is-younger-than-second relation

Properties of binary relations

- A relation \( R \) over set \( S \) is reflexive means \( R(x,x) \) for all \( x \) in \( S \)
  - e.g., the relation “\( \leq \)” on the set of integers \( \{1, 2, 3\} \) is \([\{(1,1),(1,2),(1,3),(2,2),(2,3),(3,3)\} \)
  - It is reflexive because \( (1,1), (2,2), (3,3) \) are in this relation.
- A relation \( R \) on a set \( S \) is symmetric if and only if for any \( x \) and \( y \) in \( S \), whenever \( (x,y) \) is in \( R \), \( (y,x) \) is in \( R \).
  - e.g., The relation “\( = \)” on the set of integers \( \{1, 2, 3\} \) is symmetric.
  - The relation “being acquainted with” on a set of people is symmetric.
- A binary relation \( R \) over set \( S \) is transitive means:
  - If \( R(x,y) \) and \( R(y,z) \) then \( R(x,z) \) for all \( a,b,c \) in \( S \)
  - e.g., The relation “\( \leq \)” on the set of integers \( \{1, 2, 3\} \) is transitive, because for \( (1,2) \) and \( (2,3) \) in \( \leq \), \( (1,3) \) is also in \( \leq \) (and similarly for the others)
Equivalence relations

- A binary relation $R$ is an equivalence relation if $R$ is reflexive, symmetric, and transitive.

- Examples
  - Same gender
  - Connected roads in the world
  - "Is equal to" on the set of real numbers
  - "Has the same birthday as" on the set of all people
  - ...

Punch-line

- Equivalence relations give rise to partitions.
- Every partition induces an equivalence relation.
- Every equivalence relation induces a partition.

- Suppose $P = \{S_1, S_2, ..., S_n\}$ is a partition.
  - Define $R(x, y)$ to mean $x$ and $y$ are in the same $S_i$.
  - $R$ is an equivalence relation.

- Suppose $R$ is an equivalence relation over $S$.
  - Consider a set of sets $S_1, S_2, ..., S_n$ where
    (1) $x$ and $y$ are in the same $S_i$ if and only if $R(x, y)$
    (2) Every $x$ is in some $S_i$.
  - This set of sets is a partition.

Example

- Let $S$ be $\{a, b, c, d, e\}$
- One partition: $\{a, b, c\}$, $\{d\}$, $\{e\}$
- The corresponding equivalence relation: $(a, a)$, $(b, b)$, $(c, c)$, $(a, b)$, $(b, a)$, $(a, c)$, $(c, a)$, $(b, c)$, $(c, b)$, $(d, d)$, $(e, e)$

The Union-Find ADT

- The union-find ADT (or "Disjoint Sets" or "Dynamic Equivalence Relation") keeps track of a set of elements partitioned into a number of disjoint subsets.

- Many uses (which is why an ADT taught in CSE 373):
  - Road/network/graph connectivity (will see this again)
  - "connected components" e.g., in social network
  - Partition an image by connected-pixels-of-similar-color (possible optional programming problem)
  - Type inference in programming languages

- Not as common as dictionaries, queues, and stacks, but valuable because implementations are very fast, so when applicable can provide big improvements.

Connected Components of an Image

- Gray tone image
- Binary image
- Cleaned up
- Components
Union-Find Operations

- Given an unchanging set $S$, create an initial partition of a set
  - Typically each item in its own subset: $\{a\}, \{b\}, \{c\}, \ldots$
  - Give each subset a "name" by choosing a representative element

- Operation find takes an element of $S$ and returns the representative element of the subset it is in
- Operation union takes two subsets and (permanently) makes one larger subset
  - A different partition with one fewer set
  - Affects result of subsequent find operations
  - Choice of representative element up to implementation

Example

- Let $S = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- Let initial partition be (will highlight representative elements red)
  \[
  \{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}, \{7\}, \{8\}, \{9\}
  \]
- union$(2, 5)$:
  \[
  \{1\}, \{2, 5\}, \{3\}, \{4\}, \{7\}, \{8\}, \{9\}
  \]
- find$(4) = 4$, find$(2) = 2$, find$(5) = 2$
- union$(4, 6)$, union$(2, 7)$
  \[
  \{1\}, \{2, 4, 5, 6, 7\}, \{3\}, \{8\}, \{9\}
  \]
- find$(4) = 6$, find$(2) = 2$, find$(5) = 2$
- union$(2, 6)$
  \[
  \{1\}, \{2, 4, 5, 6, 7\}, \{3\}, \{8\}, \{9\}
  \]

No other operations

- All that can "happen" is sets get unioned
  - No "un-union" or "create new set" or …
- As always: trade-offs
  - Implementations will exploit this small ADT
- Surprisingly useful ADT
  - But not as common as dictionaries or priority queues

Example application: maze-building

- Build a random maze by erasing edges
  \[
  \begin{array}{cccccccc}
  \text{Start} & & & & & & & \text{End} \\
  \hline
  & & & & & & & \\
  & & & & & & & \\
  & & & & & & & \\
  \end{array}
  \]
  - Possible to get from anywhere to anywhere
    - Including "start" to "finish"
    - No loops possible without backtracking
    - After a "bad turn" have to "undo"

Maze building

Pick start edge and end edge

Repeatedly pick random edges to delete

One approach: just keep deleting random edges until you can get from start to finish

\[
\begin{array}{cccccccc}
\text{Start} & & & & & & & \text{End} \\
\hline
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
\end{array}
\]
Problems with this approach

1. How can you tell when there is a path from start to finish?
   - We do not really have an algorithm yet
2. We could have cycles, which a "good" maze avoids
   - Want one solution and no cycles

Revised approach

- Consider edges in random order (i.e. pick an edge)
- Only delete an edge if it introduces no cycles (how? TBD)
- When done, we will have a way to get from any place to any other place (including from start to end points)

Cells and edges

- Let's number each cell
  - 36 total for 6 x 6
- An (internal) edge (x,y) is the line between cells x and y
  - 60 total for 6x6: (1,2), (2,3), …, (1,7), (2,8), …

The trick

- Partition the cells into disjoint sets
  - Two cells in same set if they are "connected"
  - Initially every cell is in its own subset
- If removing an edge would connect two different subsets:
  - then remove the edge and union the subsets
  - else leave the edge because removing it makes a cycle

Example

- P = disjoint sets of connected cells
- E = set of edges not yet processed, initially all (internal) edges
- M = set of edges kept in maze (initially empty)

The algorithm

```
P = disjoint sets of connected cells
E = set of edges not yet processed, initially all (internal) edges
M = set of edges kept in maze (initially empty)
while P has more than one set {
  - Pick a random edge (x,y) to remove from E
  - u = find(x)
  - v = find(y)
  - if u=v
      add (x,y) to M if same subset, do not remove edge, do not create cycle
    else
      union(u,v) if connect subsets, do not put edge in M
}
Add remaining members of E to M, then output M as the maze
```
### At the end

- Stop when `P` has one set (i.e., all cells connected)
- Suppose green edges are already in `M` and black edges were not yet picked
  - Add all black edges to `M`

#### A data structure for the union-find ADT

- Start with an initial partition of `n` subsets
  - Often 1-element sets, e.g., `{1}`, `{2}`, `{3}`, ..., `{n}`
- May have any number of `find` operations
- May have up to `n-1` `union` operations in any order
  - After `n-1` `union` operations, every `find` returns same 1 set

### Teaser: the up-tree data structure

- Tree structure with:
  - No limit on branching factor
  - References from children to parent
- Start with forest of 1-node trees
- Possible forest after several unions:
  - Will use roots for set names