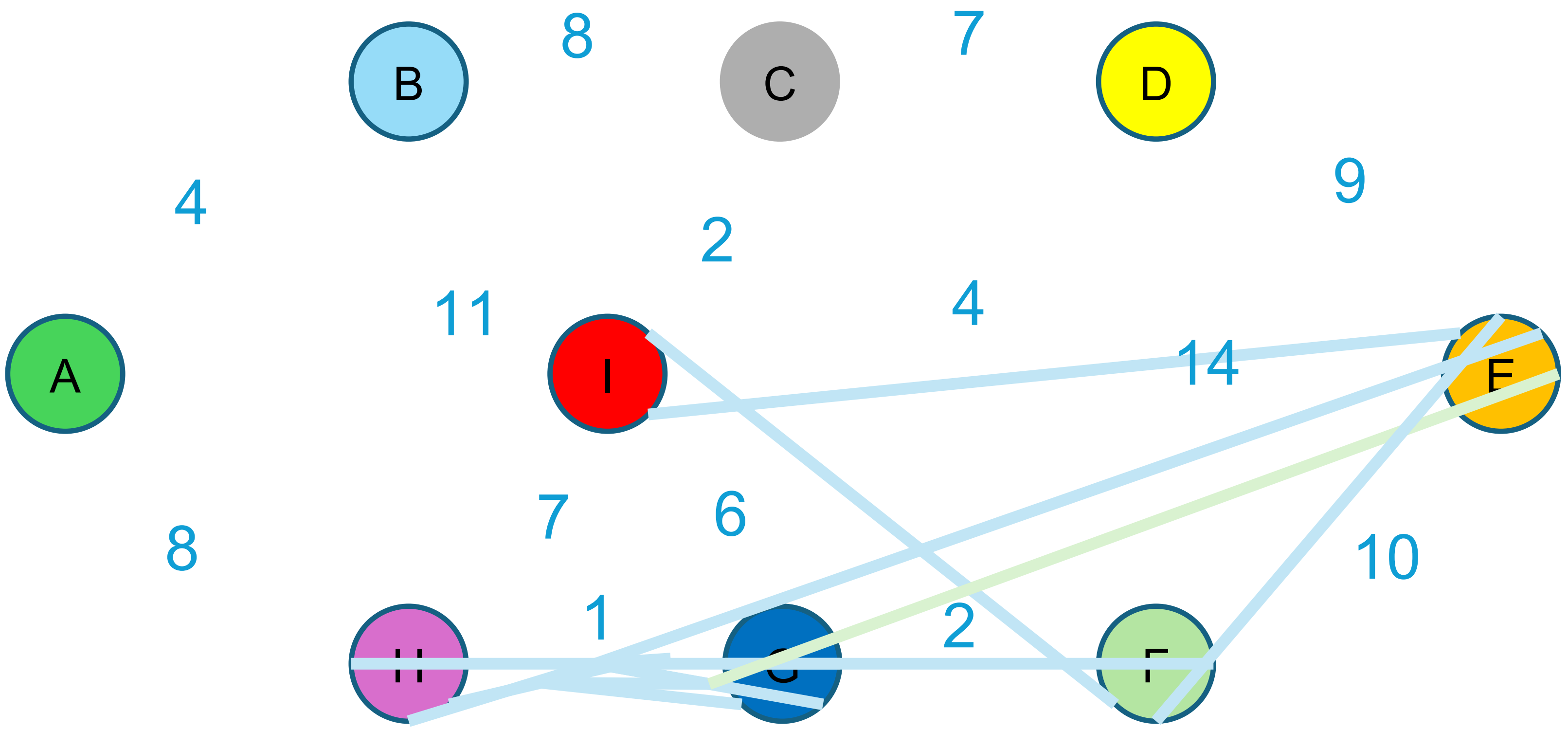
Disjoint Set Operations

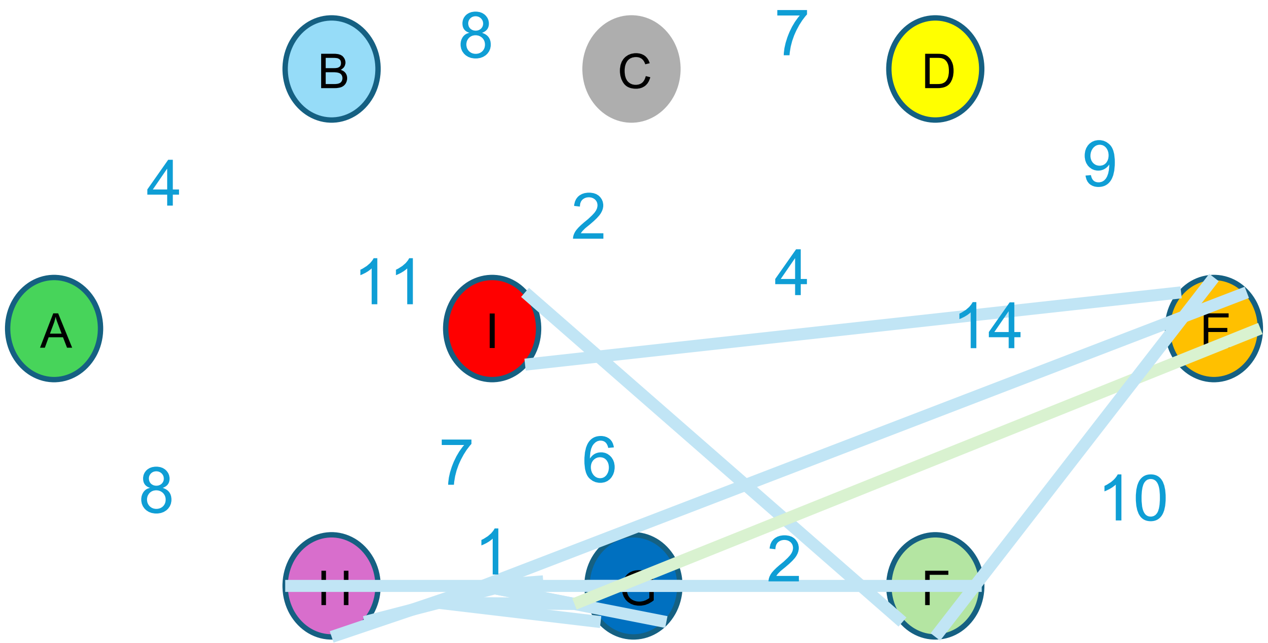
Kruskal’s Algorithm



Union-Find

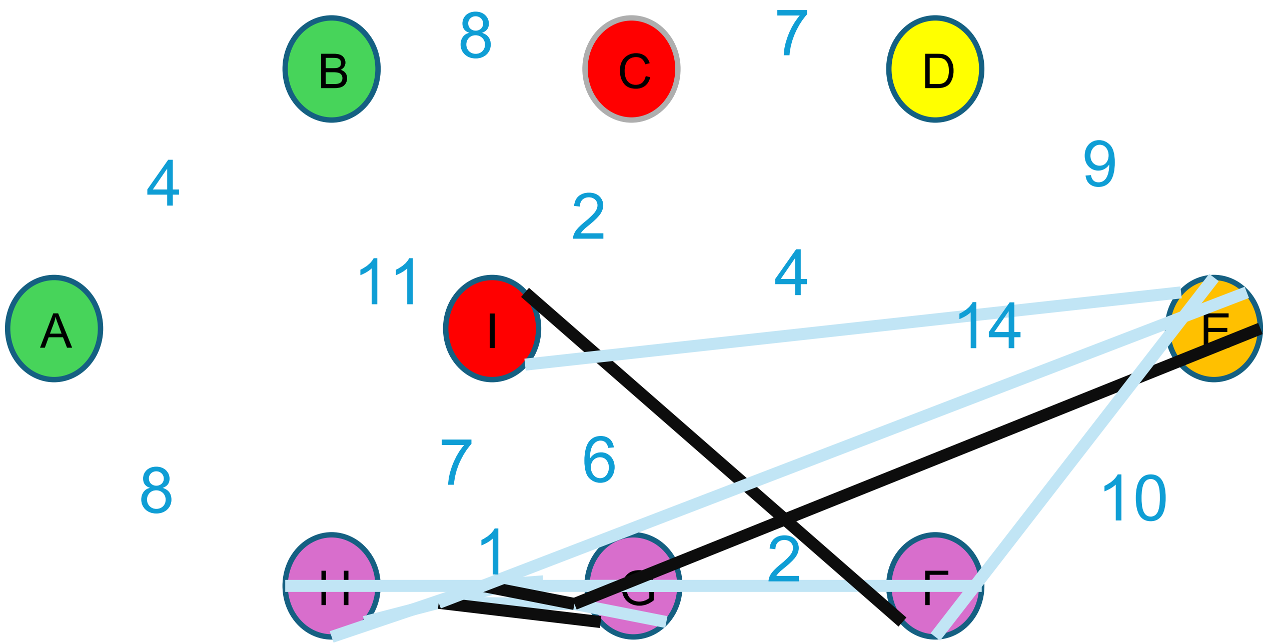
- Make-Set(x)
  - Creates a new set whose only member is x
  - O(1)

- MakeUnionFind(S)
  - MakeSet(x) with each v \in V



Union-Find

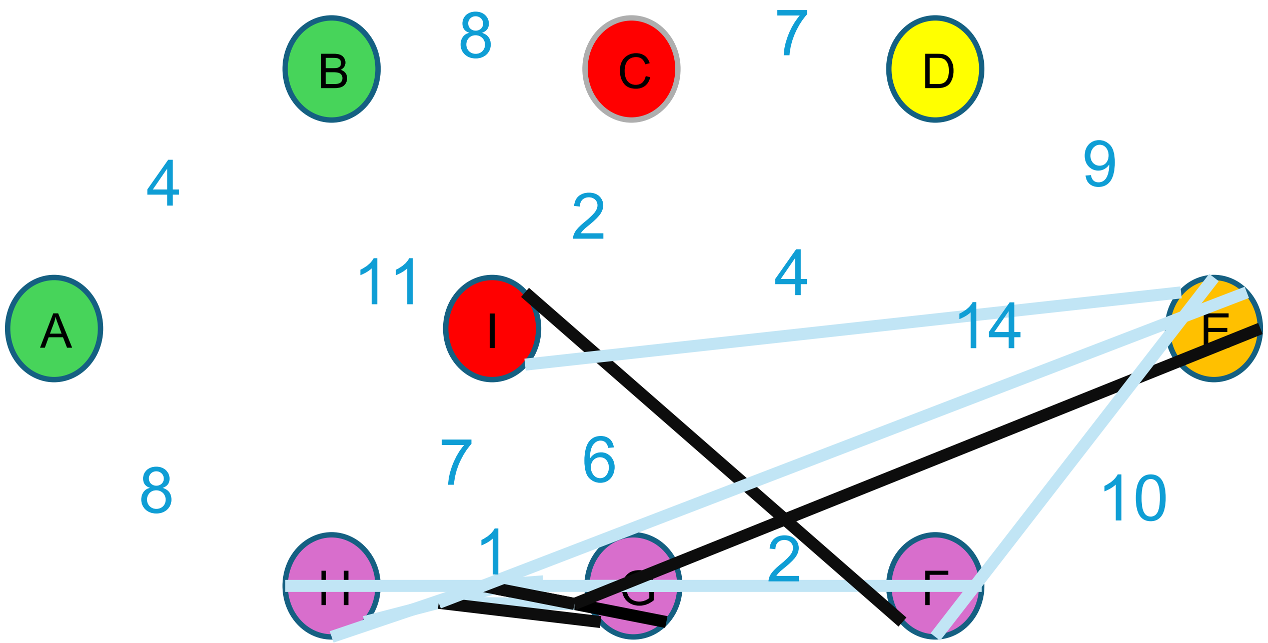
- `Find(x)`
  - Return a pointer to the representative/name of the set containing `x`
  - `Find(C)` = Red/ Pointer to C or I



Union-Find

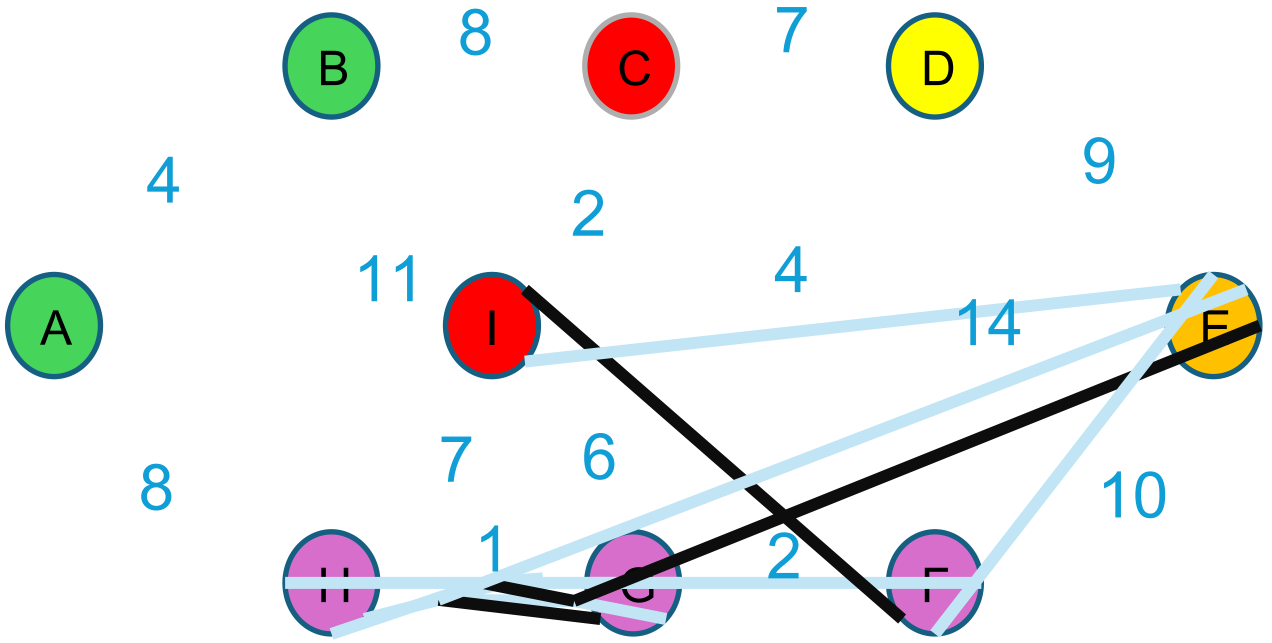
- `Find(x)`
  - Return a pointer to the representative/name of the set containing `x`
  - `Find(C)` = Red/ Pointer to C or I

- `Union(x, y)`
  - Unites two disjoint, dynamic sets that contain `x` and `y`, say S_x and S_y



Union-Find

- Maintain an array `Component`
  - Contains the name of the set currently containing each element.
- Name of the set
  - Pointer to a representative node
  - Name of the representative node

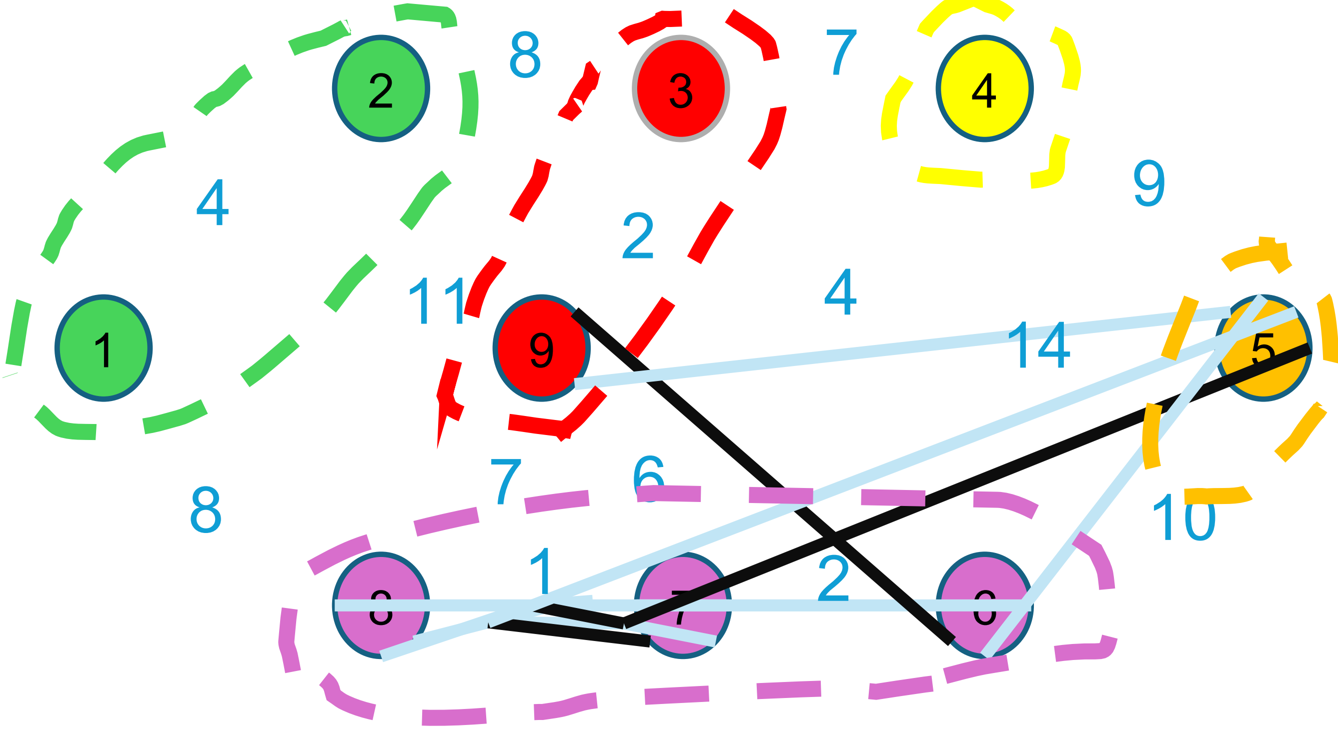


Union-Find

- Maintain an array `Component`
  - Contains the name of the set currently containing each element.
- Name of the set
  - Pointer to a representative node
  - Name of the representative node
  - Index of the representative node

Components

1	1	3	4	5	6	6	6	3
1	2	3	4	5	6	7	8	9

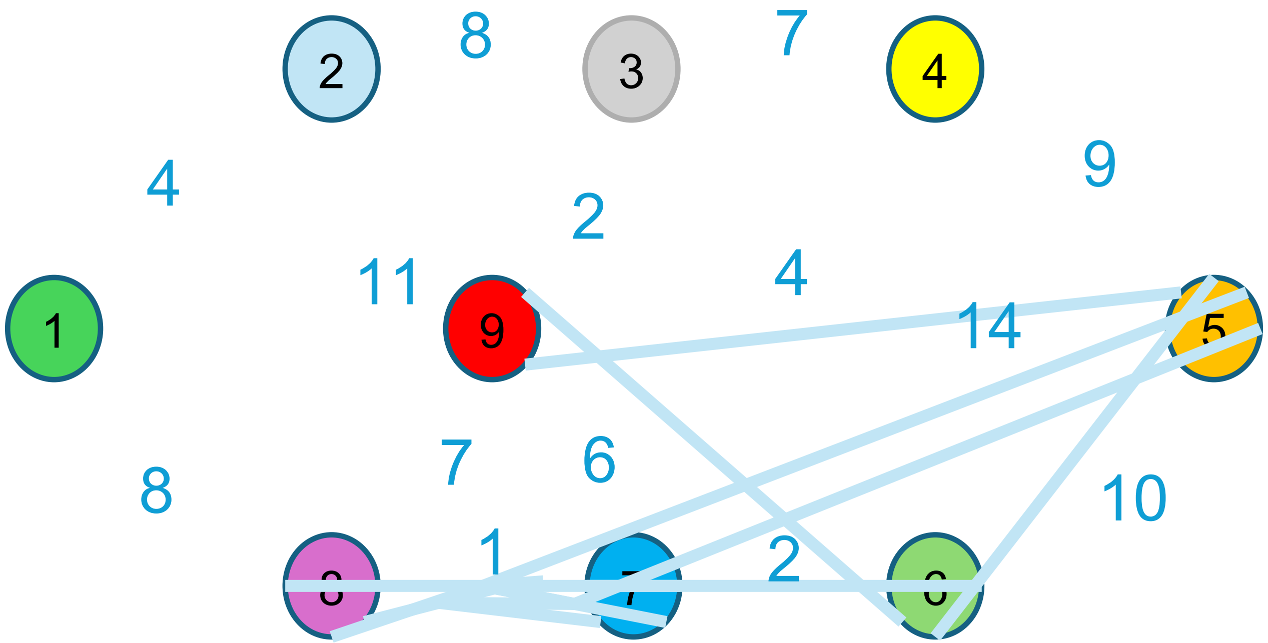


Union-Find

- Initially set $Component[s] = s$ for all s.

Components

1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9



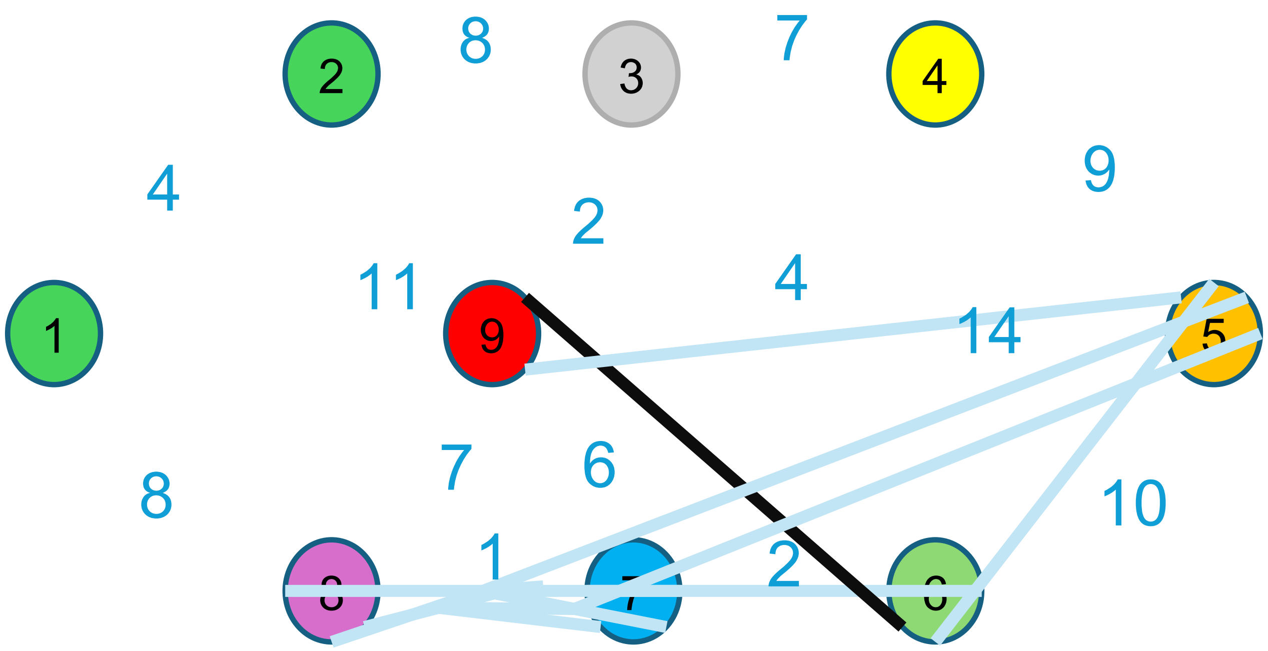
Union-Find

- Initially set $Component[s] = s$ for all `s`.

- `Union(x, y)` merges two disjoint sets together
  - Update the values of $Component[s]$ for all elements in sets A and/or B

Components

1	1	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9

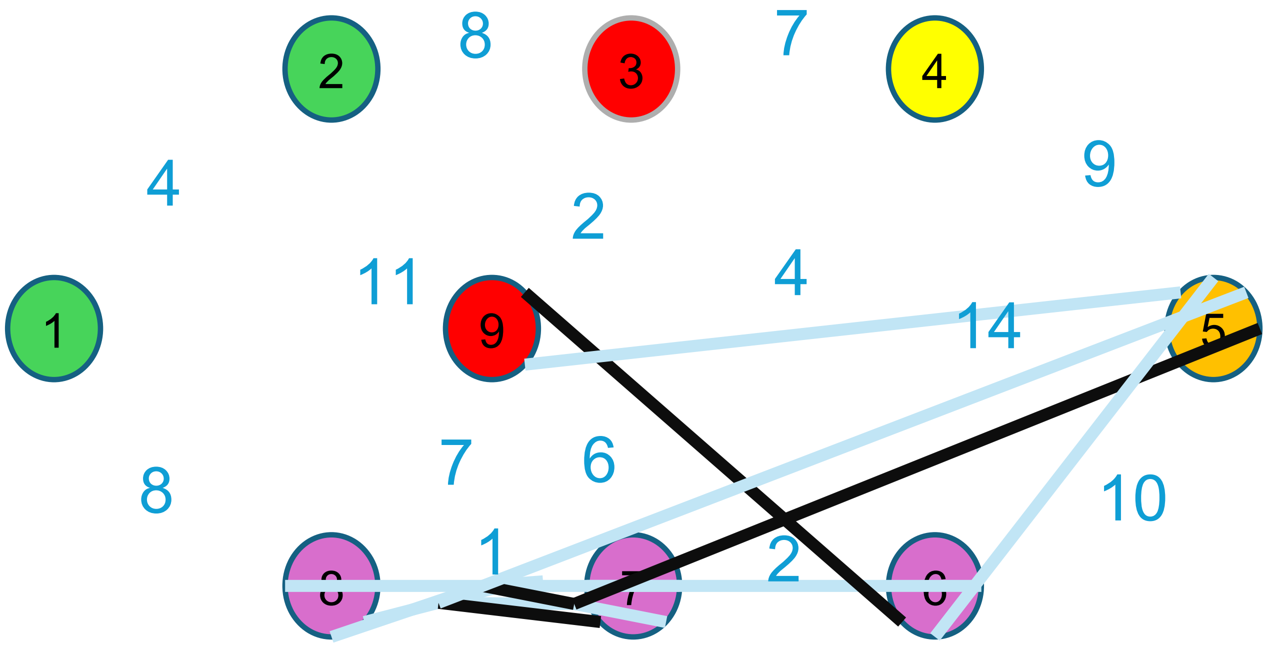


Union-Find

- Initially set Component[s]=s for all s
- Union(x, y) merges two disjoint sets together
  - Update the values of Component[s] for all elements in sets A and/or B

Components

1	1	3	4	5	6	6	6	3
1	2	3	4	5	6	7	8	9

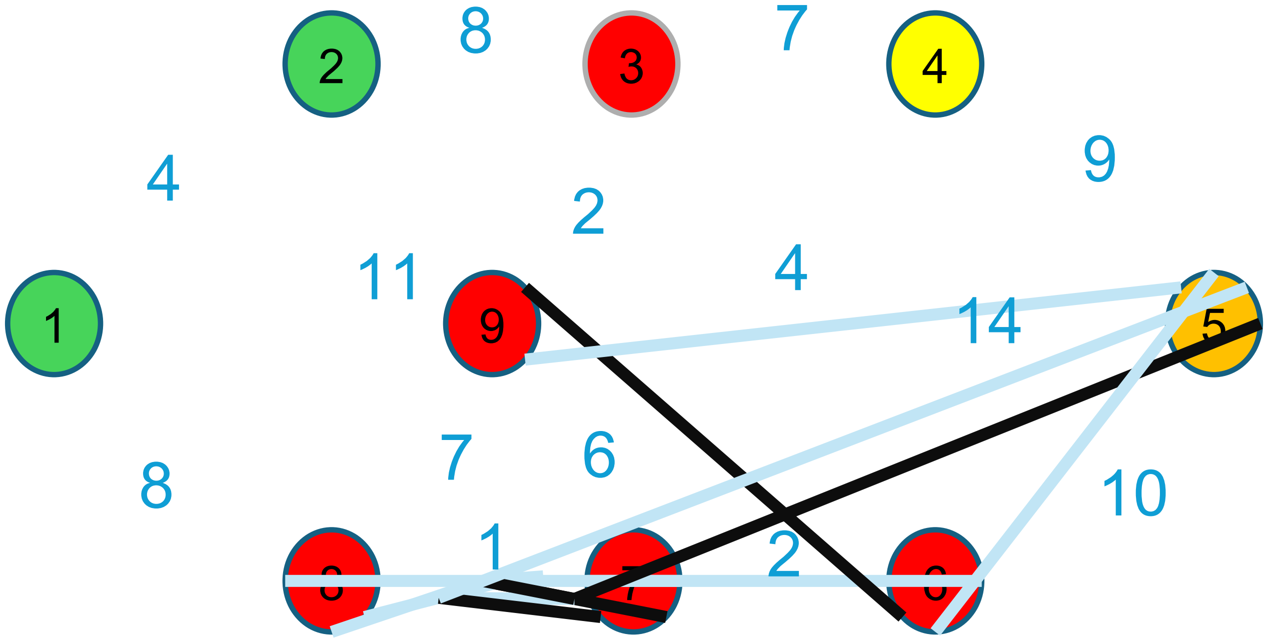


Union-Find

- Initially set Component[s]=s for all s
- Union(x, y) merges two disjoint sets together
  - Update the values of Component[s] for all elements in sets A and B
  - Scan all the components
  - Can take $O(n)$

Components

1	1	3	4	5	3	3	3	3
1	2	3	4	5	6	7	8	9

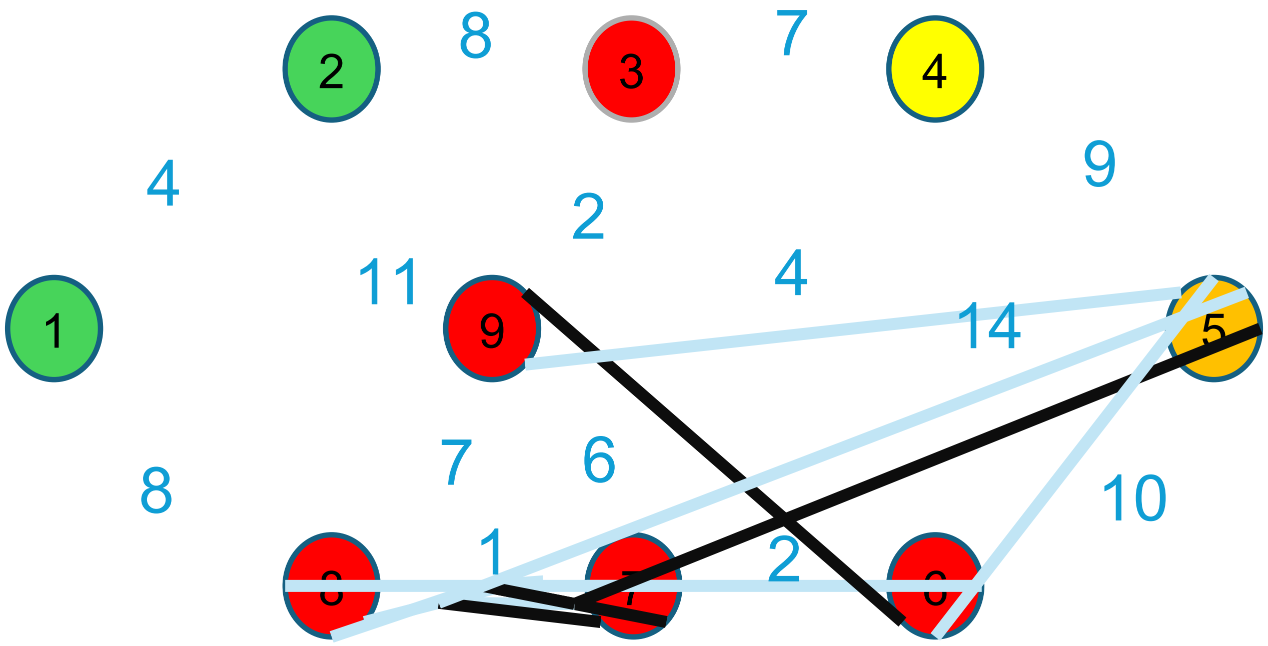


Union-Find

- Find(x)
  - Return `Components[x]`
  - Takes $O(1)$

Components

1	1	3	4	5	3	3	3	3
1	2	3	4	5	6	7	8	9

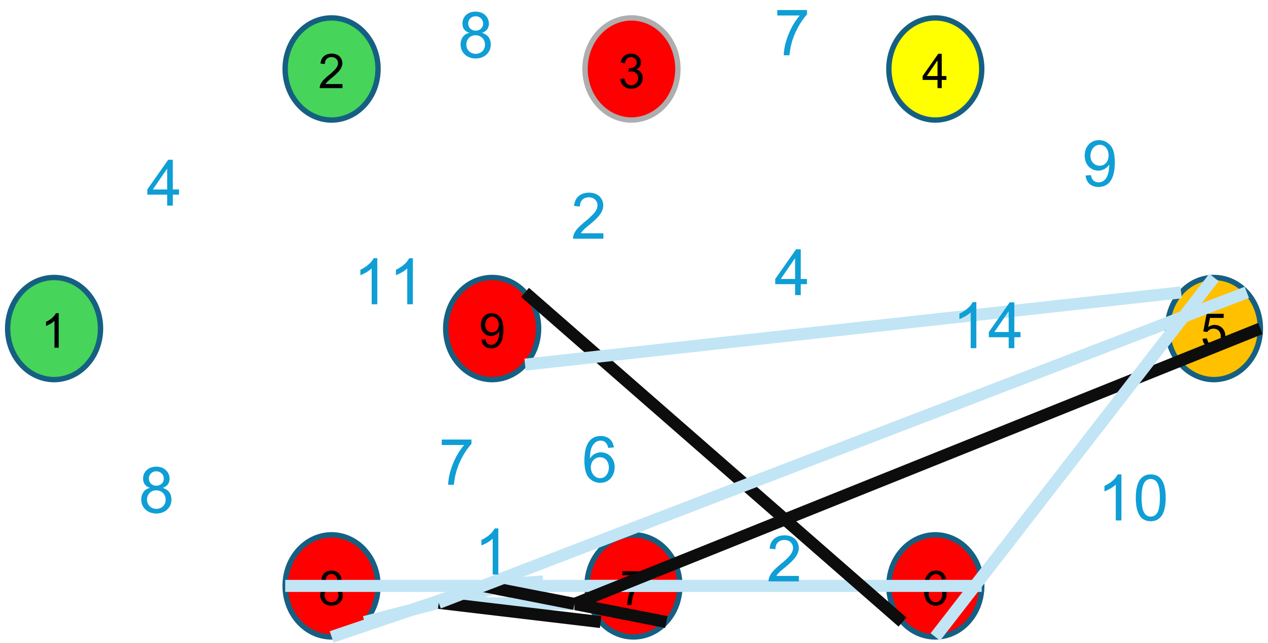


Union-Find

- Optimizations to improve the `Union(x, y)`
  - Maintain the list of elements in each component
  - Only update the elements in the smaller set; Keep the name of the larger set
  - Still $O(n)$

Components

1	1	3	4	5	3	3	3	3
1	2	3	4	5	6	7	8	9



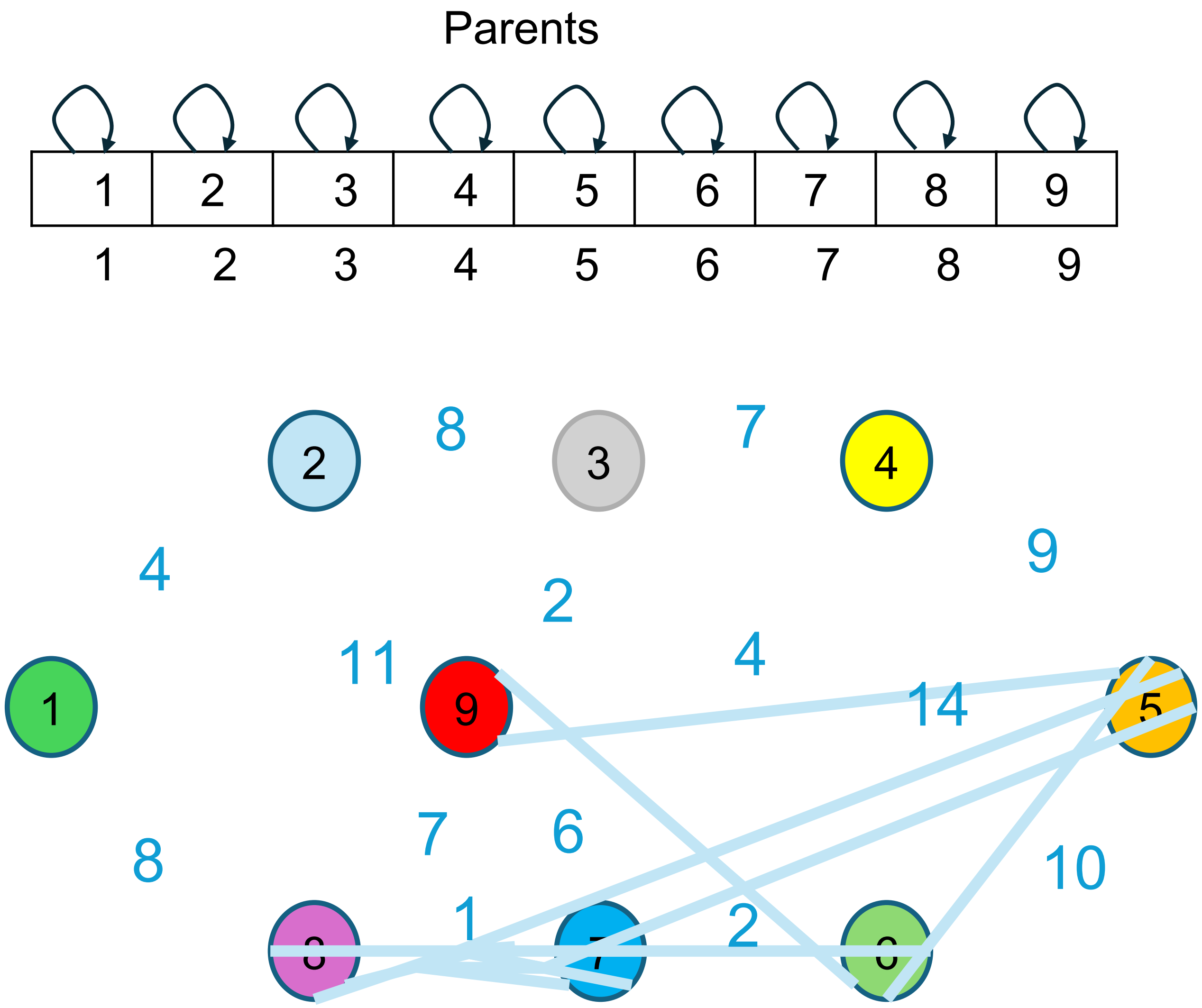
Union-Find

- Any sequence of `k` Union operations takes at most $O(k \log k)$ time
  - Touches at most 2k elements of S
  - A node v’s set grows after each Union operation
  - Either `Component[v]` remains unchanged, or it is updated
  - If updated the size of v’s set doubles
  - There can be at most $\log(2k)$ updates to `Component[v]`
  - For 2k node, there can be at most $O(klogk)$ updates.

A Better Union-Find

- Each node v will point to the representative node of its set.

- MakeUnionFind(S) initializes a record for each element v with a pointer that points to itself
  - To indicate that v is in its own set.

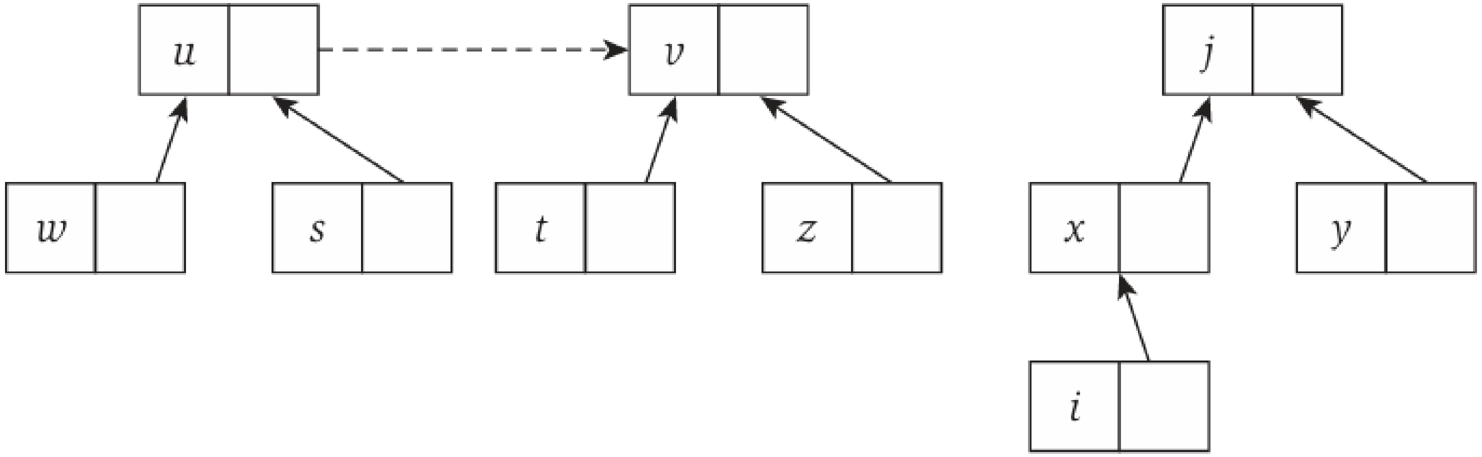


Parents

1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9

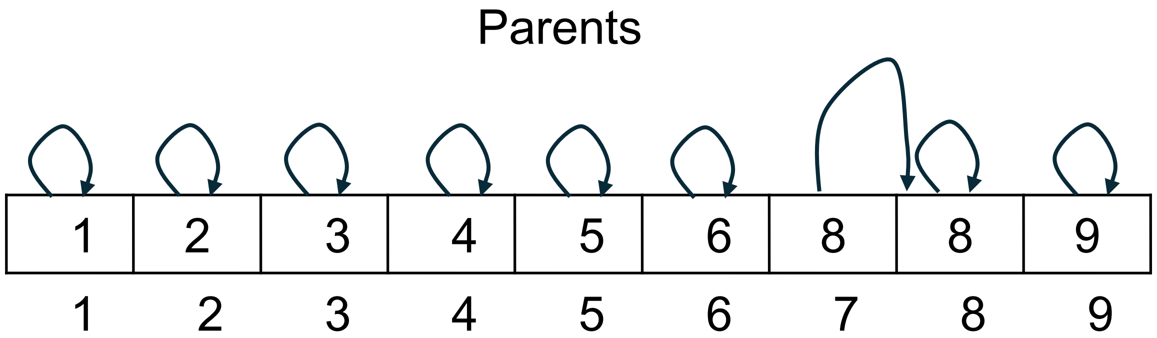
A Better Union-Find

- Consider a Union(x, y)
  - Set either x or y be the name of the combined set (preferably from the larger set)
  - Assume we select y as the name.
  - Simply update x’s pointer to point to y.
  - We do not update the pointers at the other nodes in x’s set.



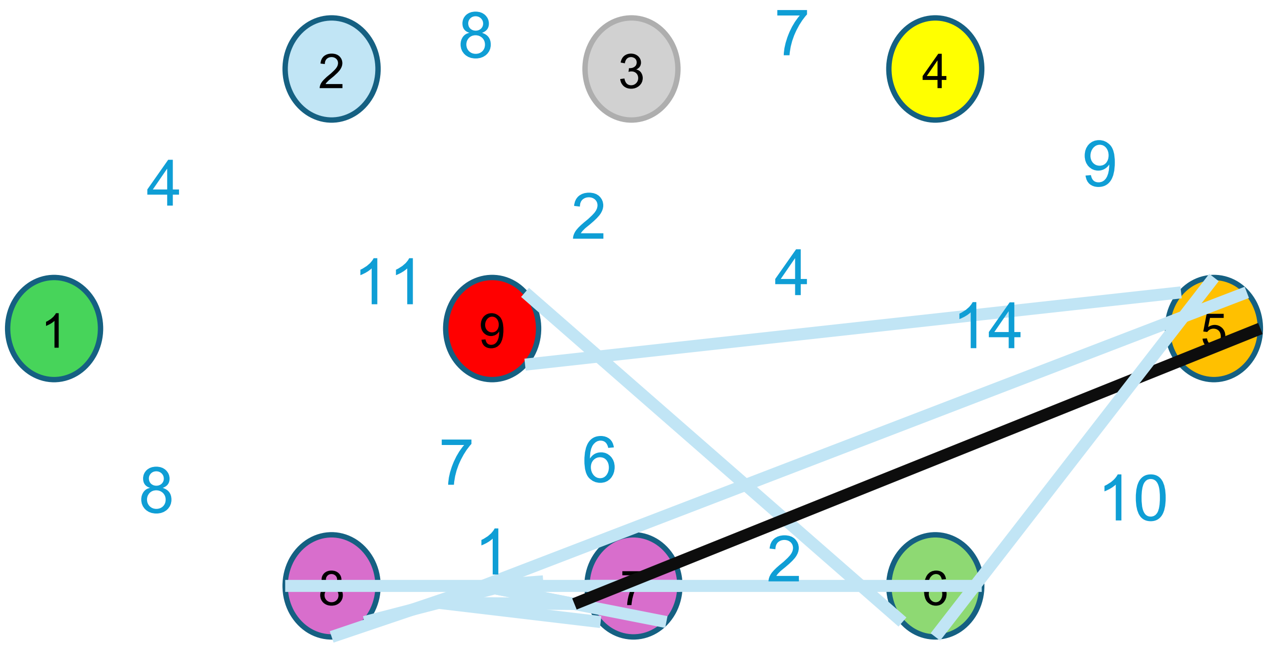
A Better Union-Find

- Consider a Union(x, y)
  - Set either x or y be the name of the combined set
  - Assume we select y as the name.
  - Simply update x’s pointer to point to y.
  - We do not update the pointers at the other nodes in x’s set.



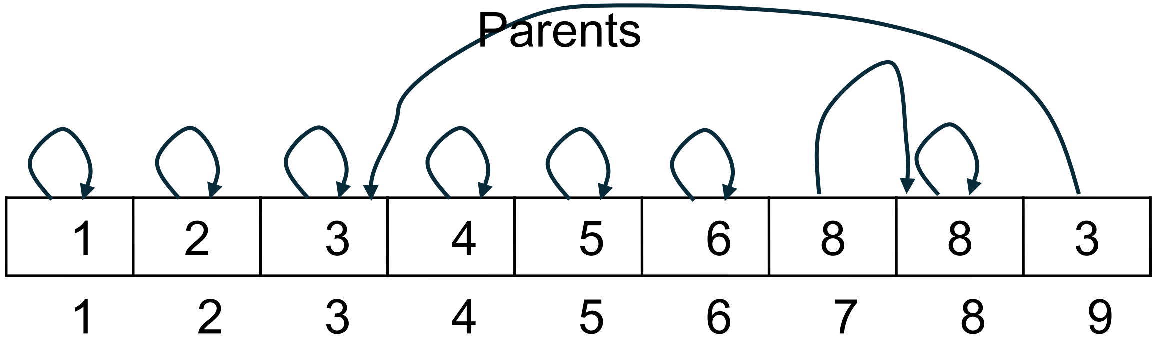
Parents

1	2	3	4	5	6	8	8	9
1	2	3	4	5	6	7	8	9



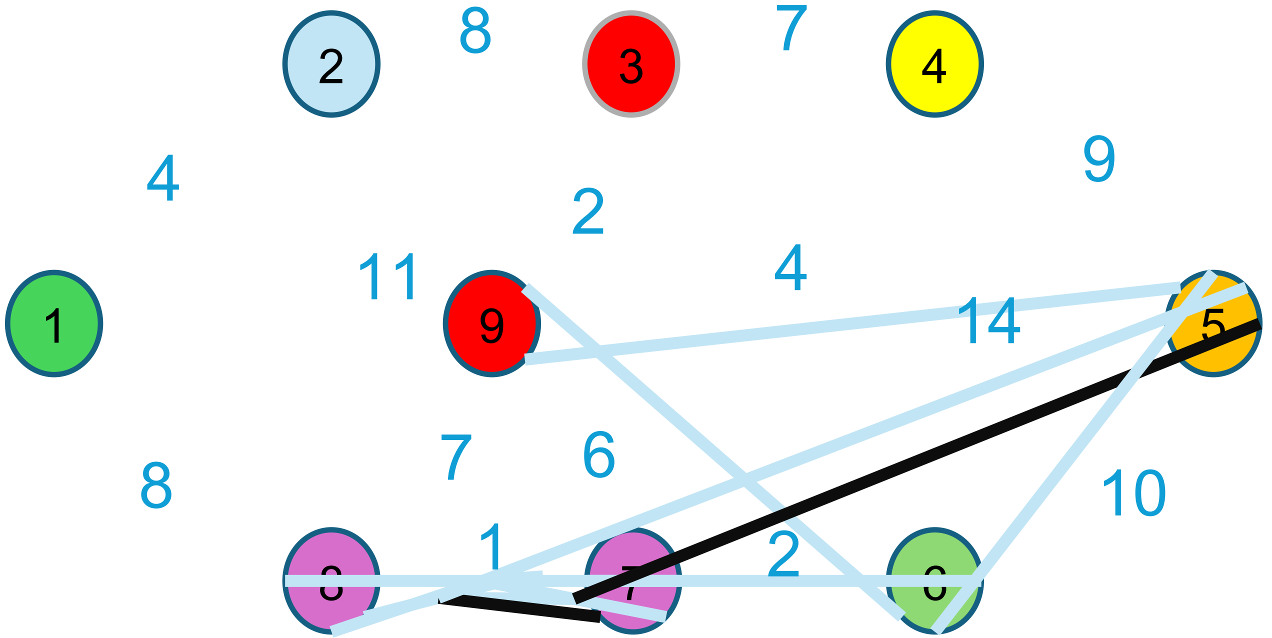
A Better Union-Find

- Consider a Union(x, y)
  - The idea is to have either x or y be the name of the combined set
  - Assume we select y as the name.
  - Simply update x’s pointer to point to y.
  - We do not update the pointers at the other nodes in x’s set.



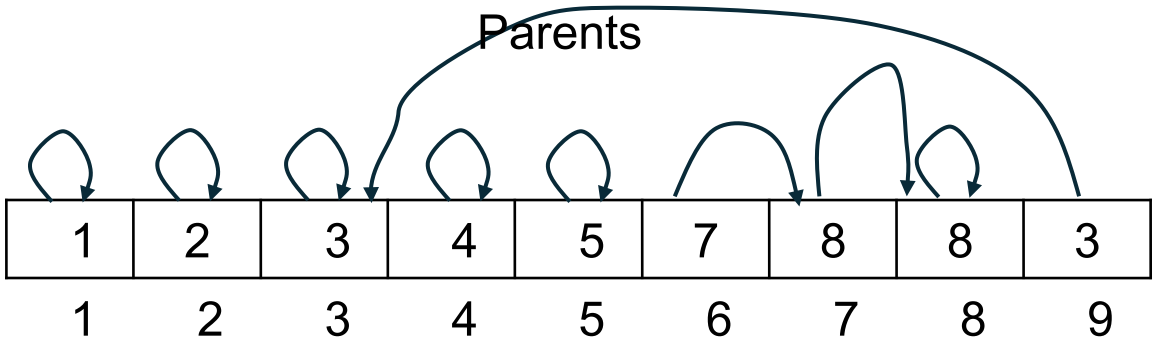
Parents

1	2	3	4	5	6	8	8	3
1	2	3	4	5	6	7	8	9



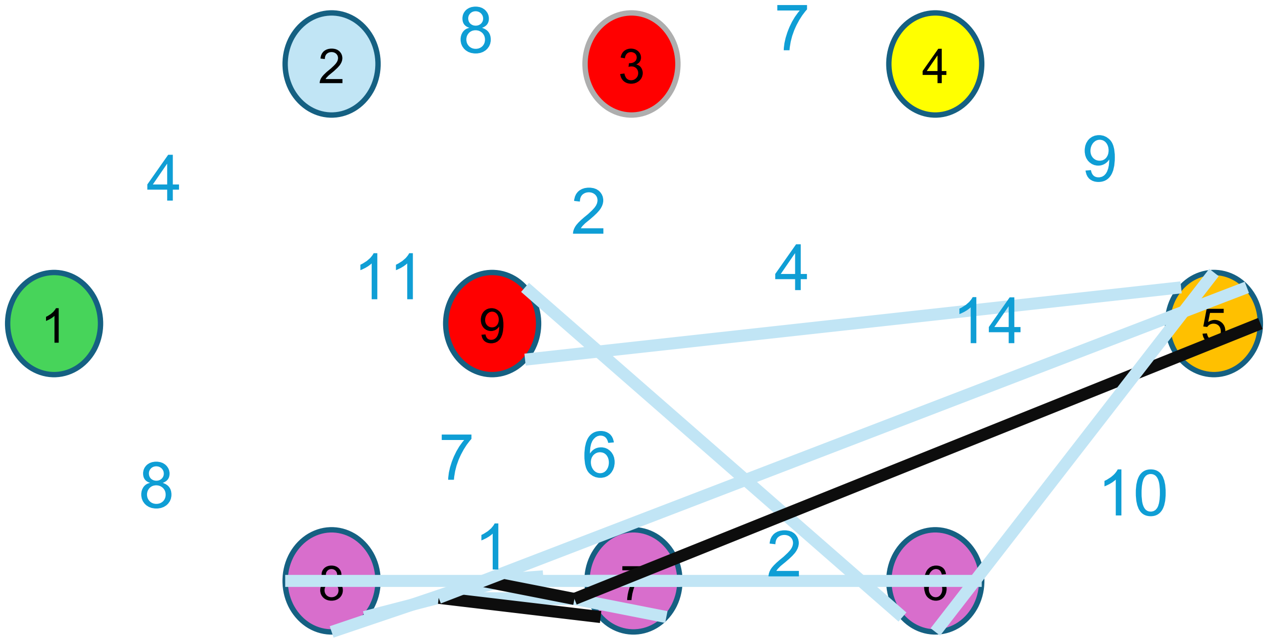
A Better Union-Find

- Consider a Union(x, y)
  - The idea is to have either x or y be the name of the combined set
  - Assume we select y as the name.
  - Simply update x's pointer to point to y.
  - We do not update the pointers at the other nodes in x's set.



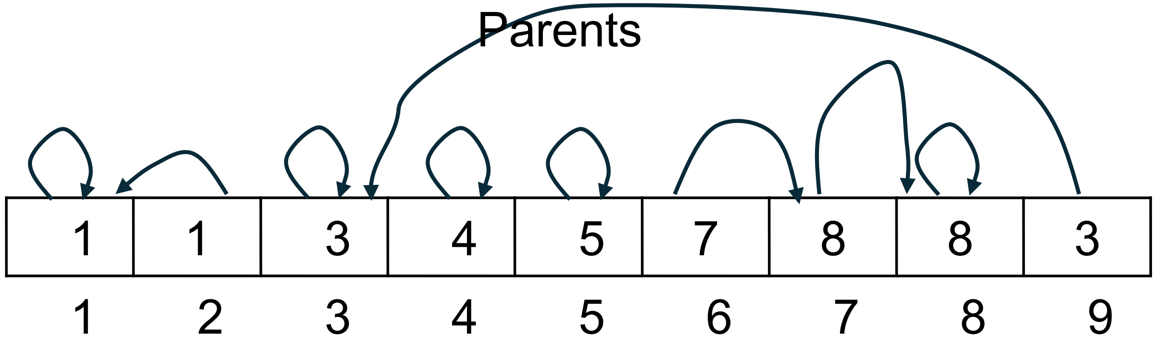
Parents

1	2	3	4	5	7	8	8	3
1	2	3	4	5	6	7	8	9



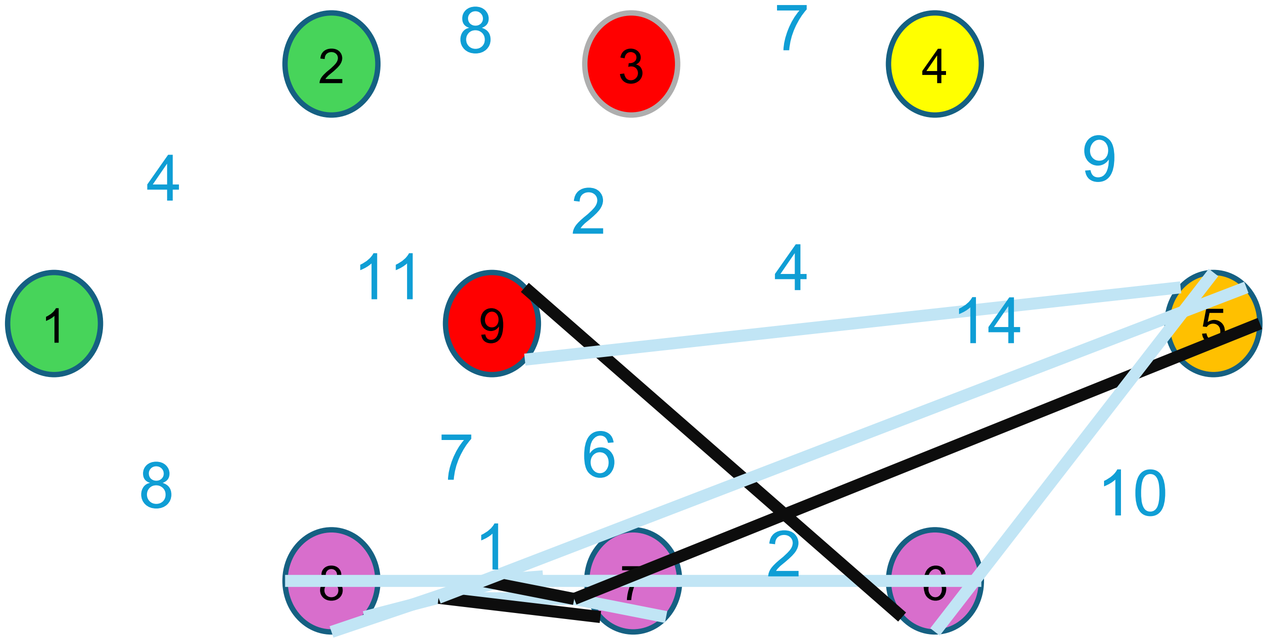
A Better Union-Find

- Consider a Union(x, y)
  - The idea is to have either x or y be the name of the combined set
  - Assume we select y as the name.
  - Simply update x's pointer to point to y.
  - We do not update the pointers at the other nodes in x's set.



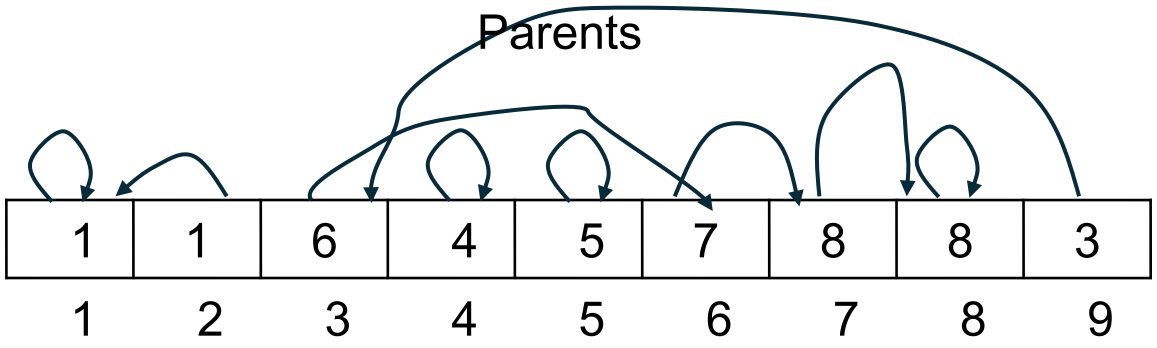
Parents

1	1	3	4	5	7	8	8	3
1	2	3	4	5	6	7	8	9



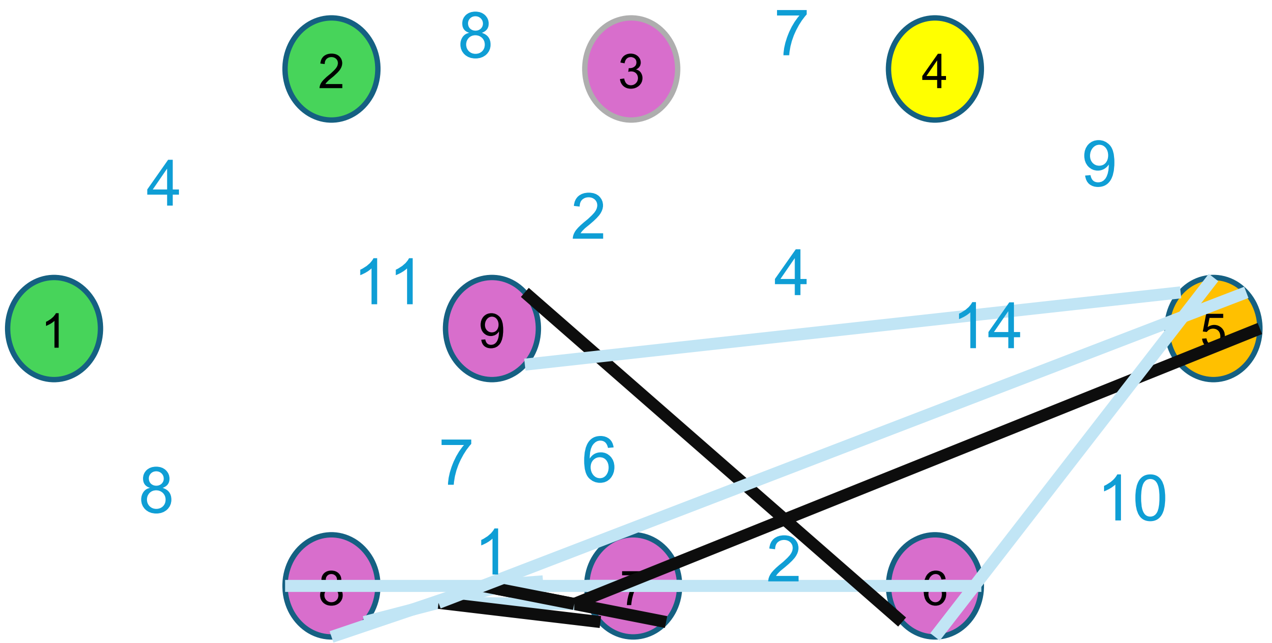
A Better Union-Find

- Consider a Union(x, y)
  - The idea is to have either x or y be the name of the combined set
  - Assume we select y as the name.
  - Simply update x's pointer to point to y.
  - We do not update the pointers at the other nodes in x's set.



Parents

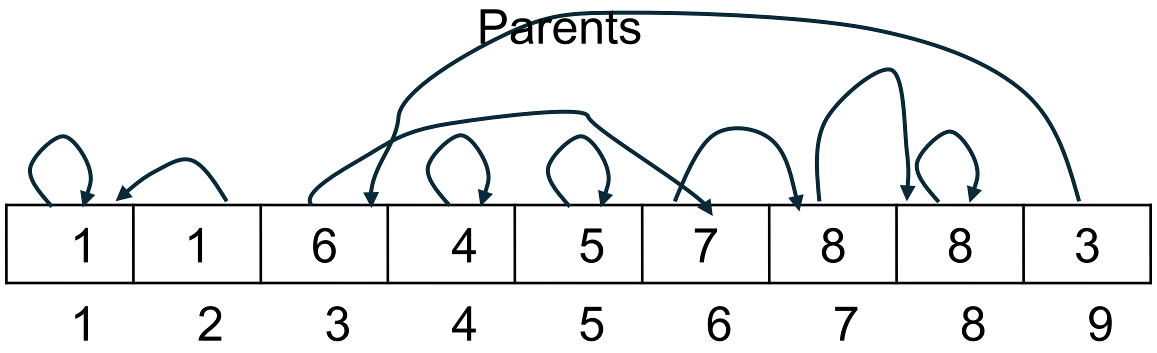
1	1	6	4	5	7	8	8	3
1	2	3	4	5	6	7	8	9



A Better Union-Find

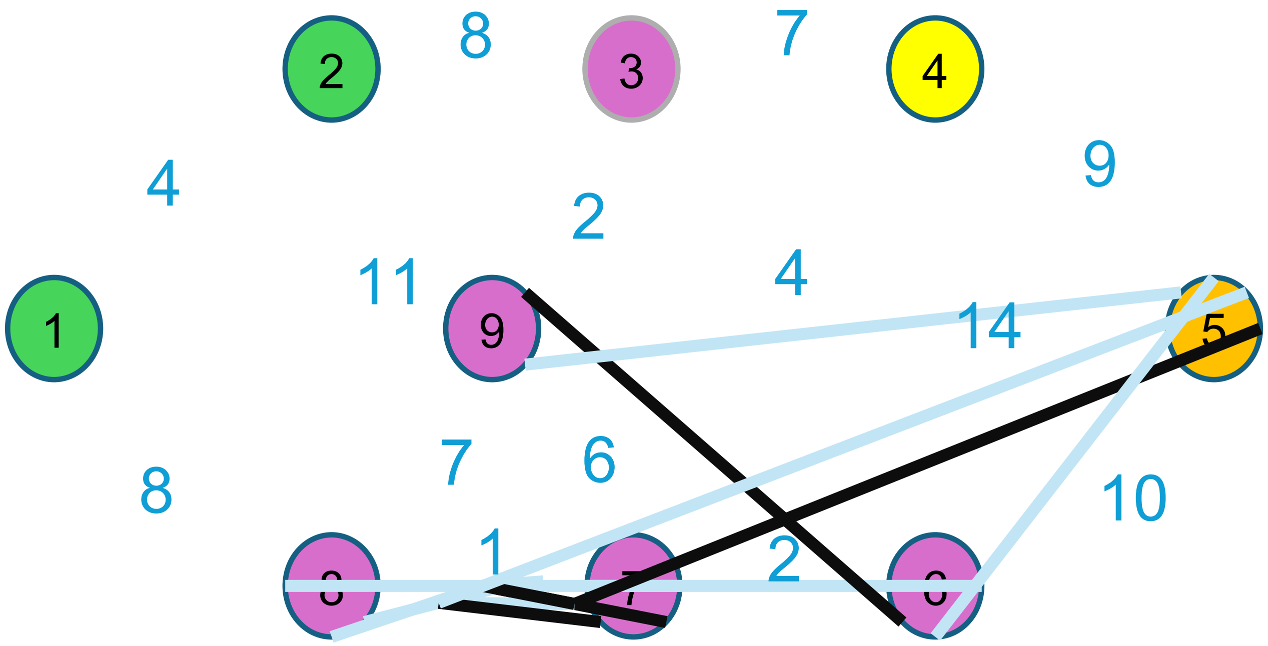
- Union(x, y)
  - Takes O(1)

- Find(x)
  - Cannot simply return Parents[s]
  - Traverse through the pointers to the top
  - No longer O(1)



Parents

1	1	6	4	5	7	8	8	3
1	2	3	4	5	6	7	8	9



A Better Union-Find

- Find operation takes O(log n) time
    - Every time the name of the set containing node v changes, the size of this set at least doubles.
    - There can be at most n nodes in a set
    - There can be at most name $\log n$ changes
    - Find operation has $O(\log n)$ complexity

A Better Union-Find

```
def MakeUnionFind(n)
   for i =  1 to n
      parent[i] = i

def Union(x, y):
   # Assuming x and y are
   # from two disjoint sets.
   if x’s set is larger
      parent[y] = x
   else
      parent[x] = y
```

```
def find(x):
   if parent[x] == x
      return parent[x]
   else
      return find(parent[x])
```

A Better Union-Find with Path Compression

```
def MakeUnionFind(n)
    for i =  1 to n
        parent[i] = i

def Union(x, y):
    # Assuming x and y are
    # from two disjoint sets.
    if x’s set is larger
        parent[y] = x
    else
        parent[x] = y
```

```
def find(x):
    if parent[x] == x
        return x
    else
        parent[x] = find(parent[x])
        return parent[x]
```

Reference

- Union-Find
  - KT Section 4.6