# Estruturas de Dados / Programação 2
## *Union-Find*

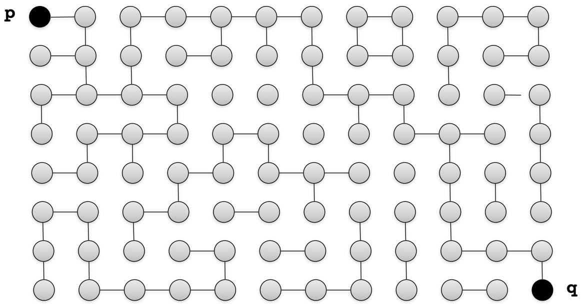
**Márcio Ribeiro**
marcio@ic.ufal.br
twitter.com/marciomribeiro

## Steps to develop an algorithm

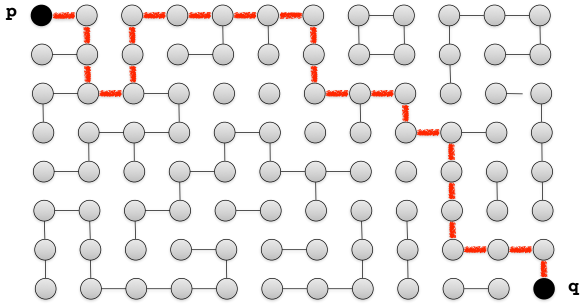
- Model the problem
- Find an algorithm to solve it
- Fast enough? Fits in memory?
- If not, figure out why
- Find a way to address the problem
- Iterate until satisfied

# Let's exercise these steps in this class!

## Network connectivity



## Network connectivity

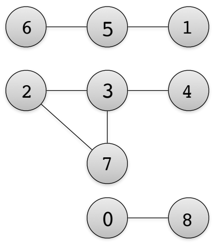


## Examples

- Variable names aliases
- Computers in a network
- Pixels in a digital photo

# Union-Find

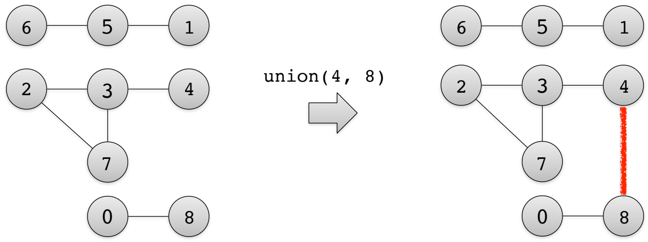
## Connected nodes



`{1 5 6} {2 3 4 7} {0 8}`

`find(4, 8) = false`

`find(4, 8) = true`

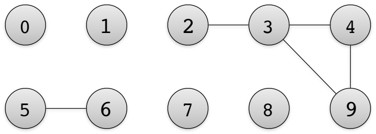


`{1 5 6} {2 3 4 7} {0 8}`

`{1 5 6} {0 2 3 4 7 8}`

# Quick find

## Data structure



| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| `id[i]` | 0 | 1 | 9 | 9 | 9 | 6 | 6 | 7 | 8 | 9 |
| `i` | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

**p** and **q** are connected if they have the same id

## How does it work?

- Find: check if p and q have the same id
- Union: to merge sets containing p and q, change all entries with id[p] to id[q]

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| `id[i]` | 0 | 1 | 9 | 9 | 9 | 6 | 6 | 7 | 8 | 9 |
| `i` | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Example: `union(3, 6)`

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| `id[i]` | 0 | 1 | 6 | 6 | 6 | 6 | 6 | 7 | 8 | 6 |
| `i` | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

# Abstract Data Type: Union-Find

## Union-Find ADT

```
void initialize();

int find(int p, int q);

void union(int p, int q);
```

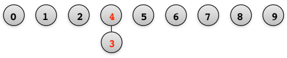
## Initialize and Find

```
void initialize()
{
  int i;
  for (i = 0; i < ARRAY_SIZE; i++) {
    id[i] = i;
  }
}

int find(int p, int q)
{
  return (id[p] == id[q]);
}
```

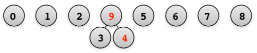
## Union

```
void union(int p, int q)
{
  int p_id = id[p];

  int i;
  for (i = 0; i < ARRAY_SIZE; i++) {
    if (id[i] == p_id) {
      id[i] = id[q];
    }
  }
}
```

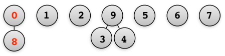
| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 3−4 | 0 | 1 | 2 | 4 | 4 | 5 | 6 | 7 | 8 | 9 |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |



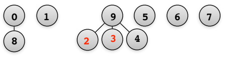
| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 4−9 | 0 | 1 | 2 | 9 | 9 | 5 | 6 | 7 | 8 | 9 |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |



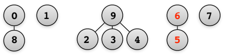
| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 8−0 | 0 | 1 | 2 | 9 | 9 | 5 | 6 | 7 | 0 | 9 |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |



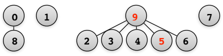
| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 2−3 | 0 | 1 | 9 | 9 | 9 | 5 | 6 | 7 | 0 | 9 |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |



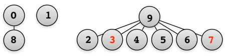
| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 5−6 | 0 | 1 | 9 | 9 | 9 | 6 | 6 | 7 | 0 | 9 |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |



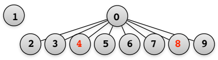
| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 5−9 | 0 | 1 | 9 | 9 | 9 | 9 | 9 | 7 | 0 | 9 |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |



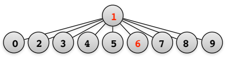
| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 7−3 | 0 | 1 | 9 | 9 | 9 | 9 | 9 | 9 | 0 | 9 |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |



| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 4−8 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |



| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 6−1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |



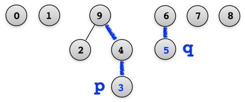
Problem: many values can change!

# Quick union

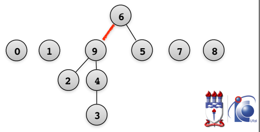
## Quick union

- Find: check if p and q have the same root
- Union: to merge sets containing p and q, set the id of q's root to the id of p's root
- id[i] is parent of i

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| id[i] | 0 | 1 | 9 | 4 | 9 | 6 | 6 | 7 | 8 | 9 |
| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |



| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| id[i] | 0 | 1 | 9 | 4 | 9 | 6 | 6 | 7 | 8 | 6 |
| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |



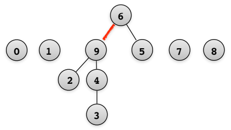
Only one value changes!

## Roots

```
int root(int i)
{
  while (i != id[i]) {
    i = id[i];
  }
  return i;
}

int find_with_roots(int p, int q)
{
  return (root(p) == root(q));
}

void union_with_roots(int p, int q)
{
  int p_root = root(p);
  int q_root = root(q);
  id[p_root] = q_root;
}
```

Any problem with this `union(3, 5)`?

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| id[i] | 0 | 1 | 9 | 4 | 9 | 6 | 6 | 7 | 8 | 6 |
| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

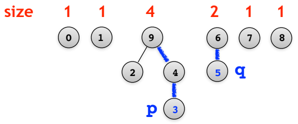


# Weighted roots

## Weighted roots

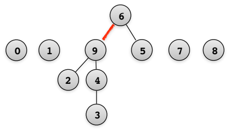
```
void union_with_weigthed_roots(int p, int q)
{
  int p_root = root(p);
  int q_root = root(q);

  if (size[p_root] < size[q_root]) {
    id[p_root] = q_root;
    size[q_root] = size[q_root] + size[p_root];
  } else {
    id[q_root] = p_root;
    size[p_root] = size[p_root] + size[q_root];
  }
}
```

| id[i] | 0 | 1 | 9 | 4 | 9 | 6 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |



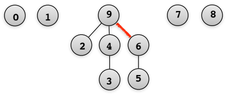
Quick union: link 9 to 6

| id[i] | 0 | 1 | 9 | 4 | 9 | 6 | 6 | 7 | 8 | 6 |
|---|---|---|---|---|---|---|---|---|---|---|
| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

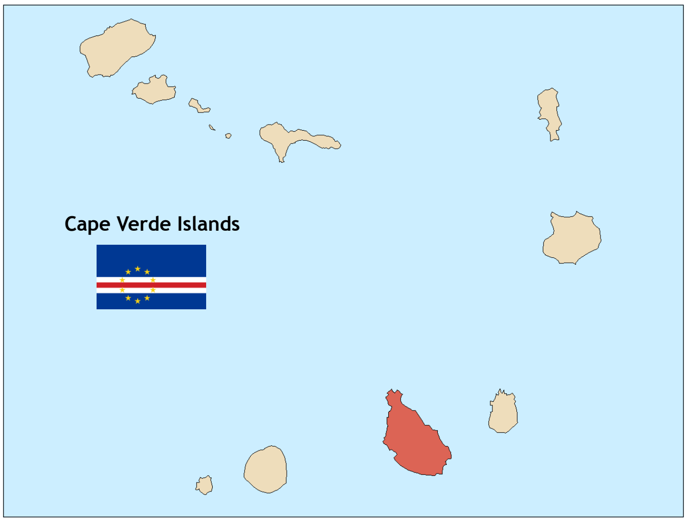


Weighted quick union: link 6 to 9

| id[i] | 0 | 1 | 9 | 4 | 9 | 6 | 9 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |



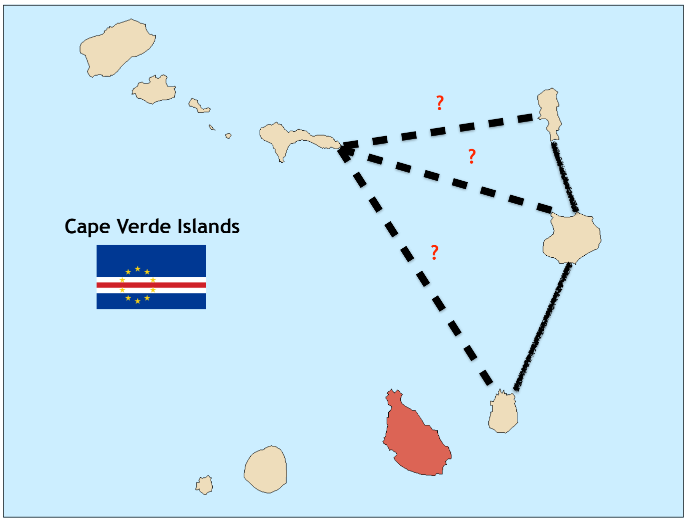
# Application



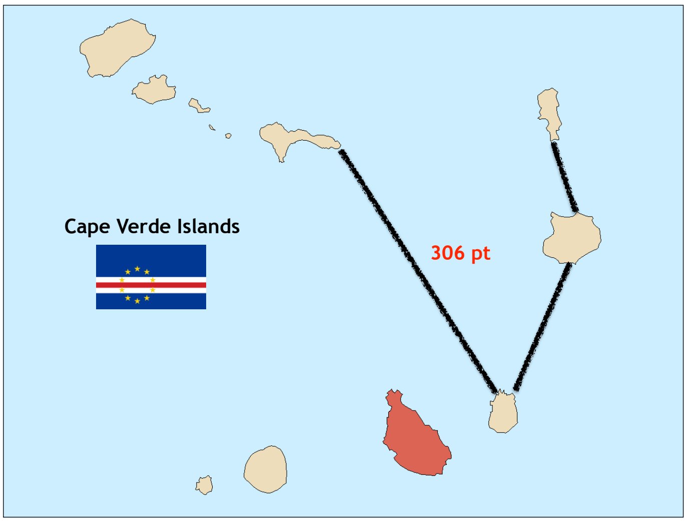
Cape Verde Islands

## Building bridges

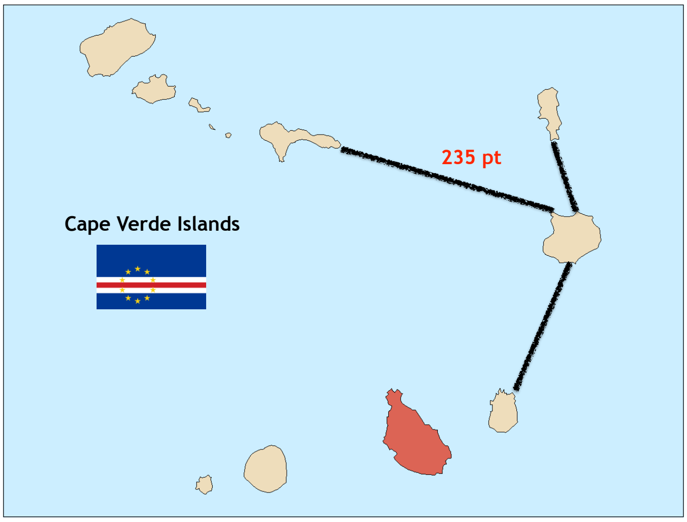
- The government wants to connect all islands
- The cost to build a bridge is proportional to the bridge’s length
- How to connect all islands with the minimum cost?

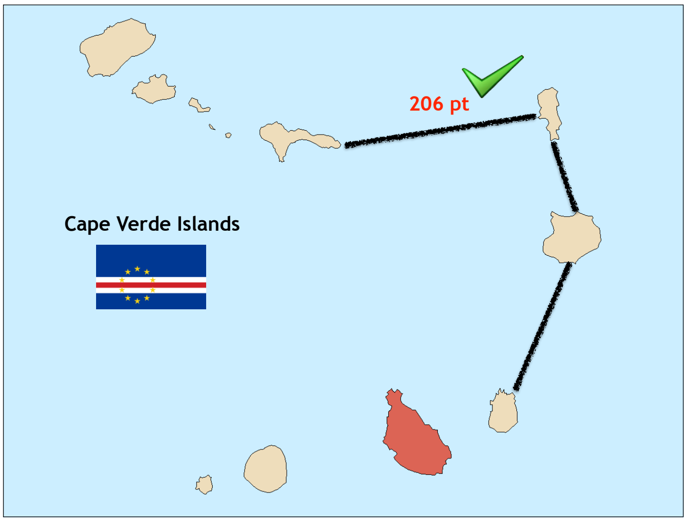


Cape Verde Islands



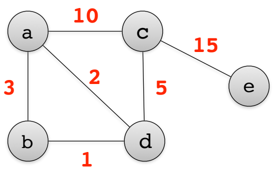
Cape Verde Islands





## Graph representation

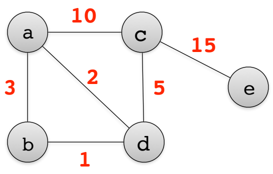
- 5 islands: the government wants to connect all
- The cost to build a bridge is proportional to the bridge's length

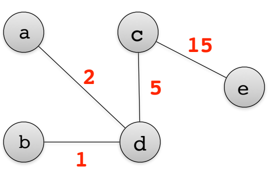


# Minimum Spanning Trees

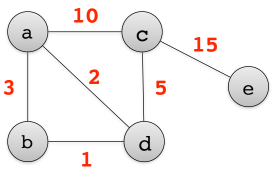
## Building bridges

- 6 islands: the government wants to connect all
- The cost to build a bridge is proportional to the bridge's length





## Kruskal Algorithm



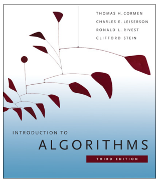
```
V = {{a}, {b}, {c}, {d}, {e}}

E = {{b,d}, {a,d}, {a,b}, {c,d}, {a,c}, {c,e}}
```

```
FIND-SET(b, d) = false
A = {} U {(b,d)} = {(b,d)}
V = {{a},{b,d},{c},{e}}

FIND-SET(a, d) = false
A = {(a,d)} U {(b,d)} = {(b,d),(a,d)}
V = {{a,b,d},{c},{e}}

FIND-SET(a, b) = true

FIND-SET(c, d) = false
A = {(c,d)} U {(b,d),(a,d)} = {(b,d),(a,d),(c,d)}
V = {{a,b,d,c},{e}}

FIND-SET(a, c) = true

FIND-SET(c, e) = false
A = {(c,e)} U {(b,d),(a,d),(c,d)} = {(b,d),(a,d),(c,d),(c,e)}
V = {{a,b,d,c,e}}
```

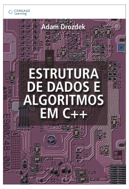
## Kruskal Algorithm

```
kruskal(G)

  A = ∅
  for each vertex v ∈ G.V
    MAKE-SET(v)

  sort(G.E)

  for each edge (p, q) ∈ G.E
    if !FIND-SET(p, q)
      A = A U {(p, q)}
      UNION(p, q)

  return A
```

## References



**Chapter 21**



**Chapter 8**