# Disjoint sets

## CS 491 – Competitive Programming

Dr. Mattox Beckman

University of Illinois at Urbana-Champaign
Department of Computer Science

Spring 2024

# Objectives

- Use the up-tree data structure to model disjoint sets
- Use path compression to make this structure efficient

# What do these groups have in common?

- Lutherans
- Jacobites
- Marxists
-

# Disjoint Sets

Properties of disjoint sets

- If $a$ is in the same set as $b$, then everything in the same set as $a$ is in the same set as $b$.
- Want to declare that $a$ and $b$ are in the same set.
- Want to check if $a$ and $b$ are in the same set.
- Want to ask how many elements are in the set.
- Want to check if all known elements are in the set.

And we want to do all of these things very very quickly.

- The datastructure for this is the up-tree.

# The idea

Representation

- Keep a vector `ds` representing all your elements.
- A positive entry means this element is in the same element as the referent.
- A negative entry means this element is the root. The magnitude is the number of elements.

Initialization

- To start, initialize the vector with all -1s.

Find operation

- `find(i) = i`, if `ds[i] < 0`
- `find(i) = find(ds[i])`, otherwise.

# Add

Relating a and b.

- Let `sa = find(a)`
- Let `sb = find(b)`

Choices:

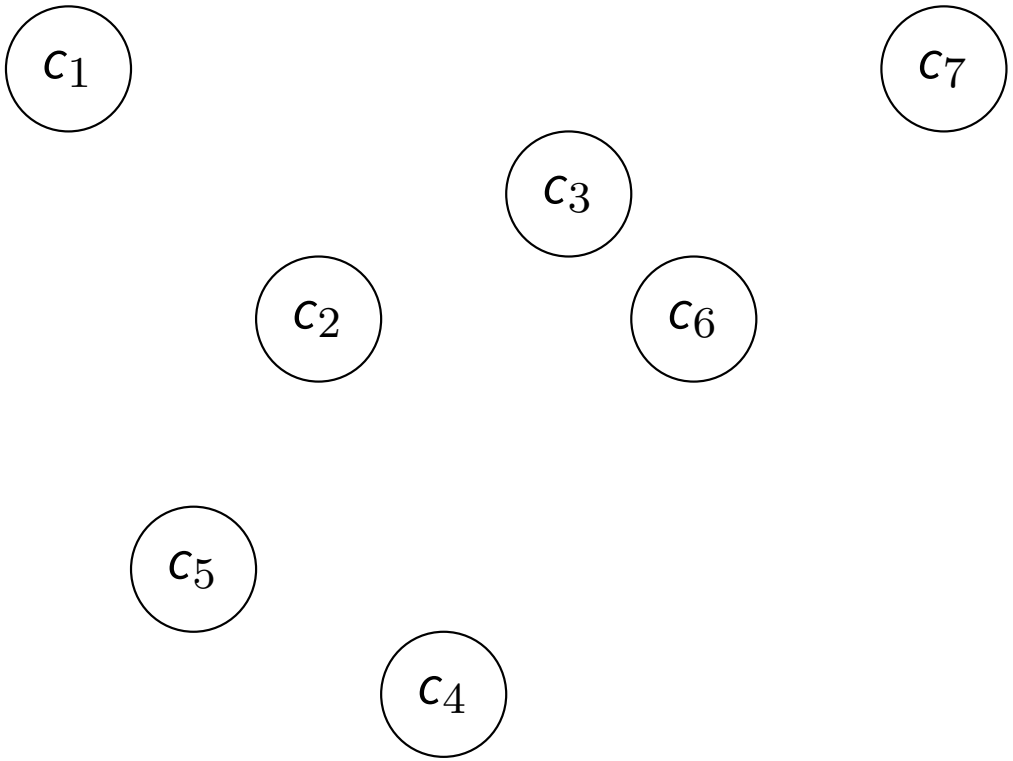
- `sa` = sb=, already in the same set.
- `ds[sa] > ds[sb]`, then `sb` is the larger set.
  - Let `ds[sb] = ds[sb] + ds[sa]` and `ds[sa] = sb`
- Symmetrically for the other case.

# Code

```
int find(int c, vi ds) {
  if (ds[c] < 0)
    return c;
  else
    return find(ds[c],ds);
}

int add(int c1, int c2, vi ds) {
  int sa = find(c1);
  int sb = find(c2);
  if (sa == sb)
    return sa;
  if (ds[sa] > ds[sb]) {
    ds[sb] += ds[sa]; ds[sa] = sb; return sb;
  } else {
    ds[sa] += ds[sb]; ds[sb] = sa; return sa;
  }
```

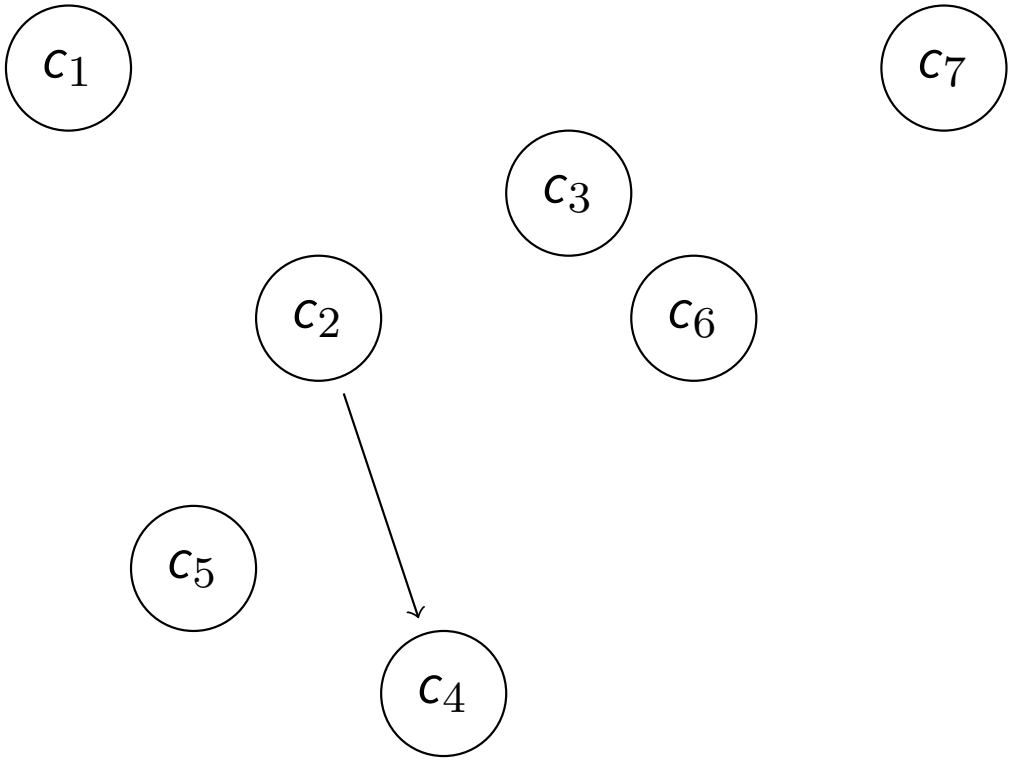
# Example, pt 1




| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| -1 | -1 | -1 | -1 | -1 | -1 | -1 |

- next connect 2 and 4

# Example, pt 2

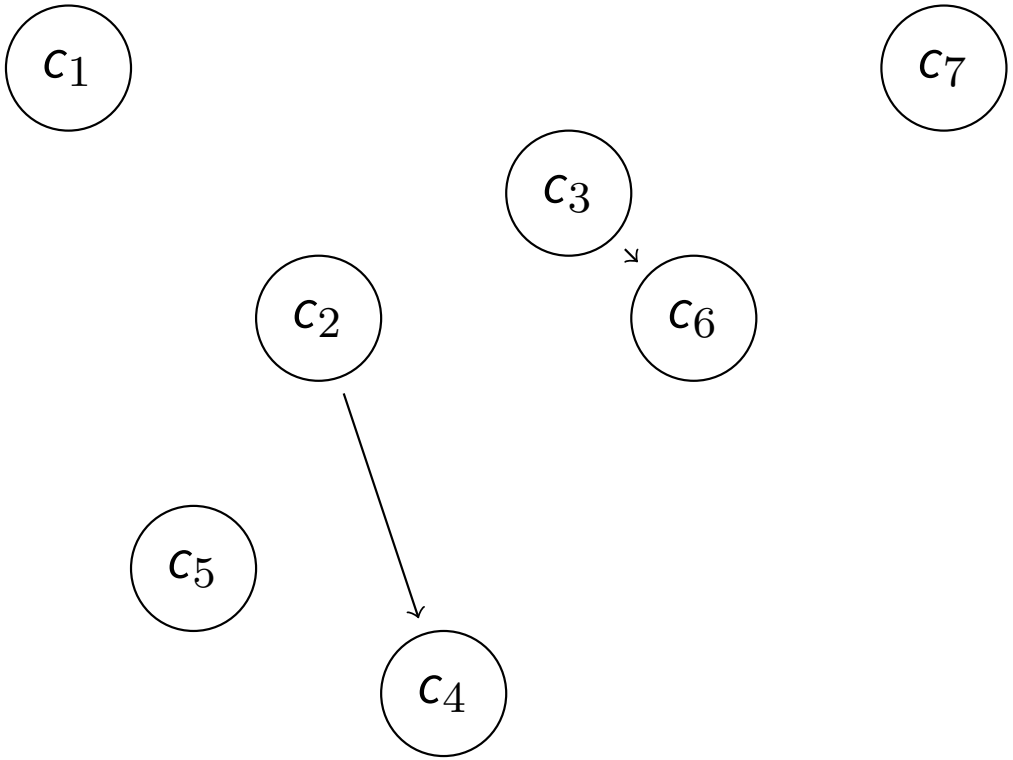



- connect 2 and 4

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| -1 | 4 | -1 | -2 | -1 | -1 | -1 |

- next connect 3 and 6

# Example, pt 3

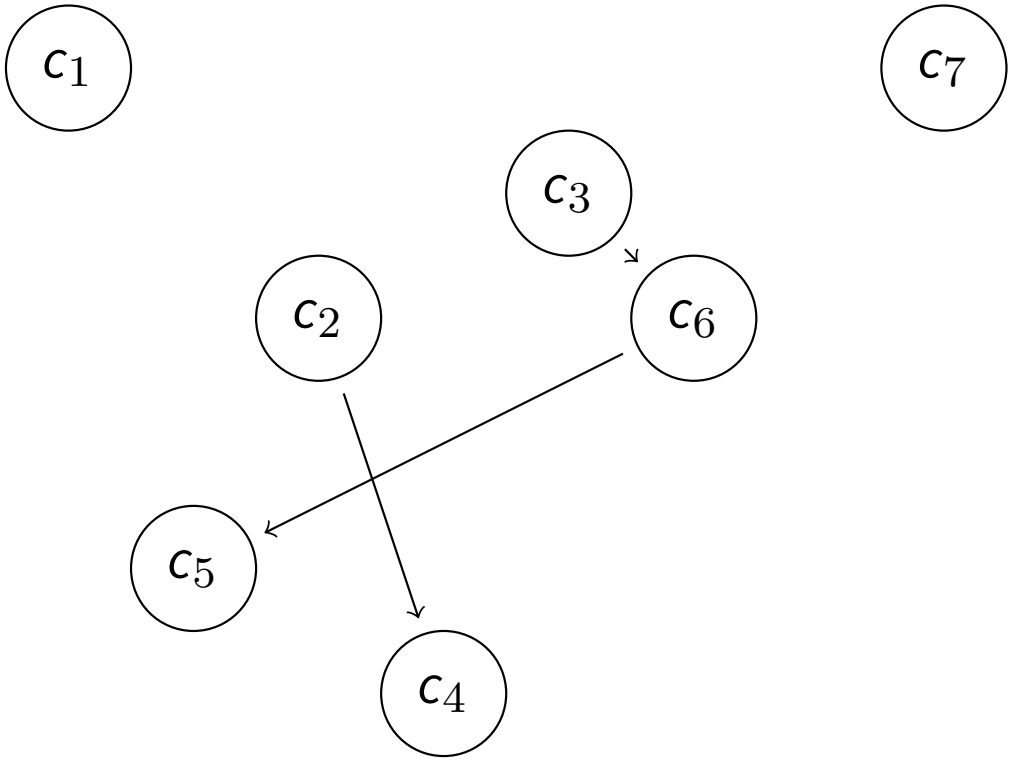



- connect 3 and 6

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| -1 | 4 | 6 | -2 | -1 | -2 | -1 |

- next connect 6 and 5

# Example, pt 4

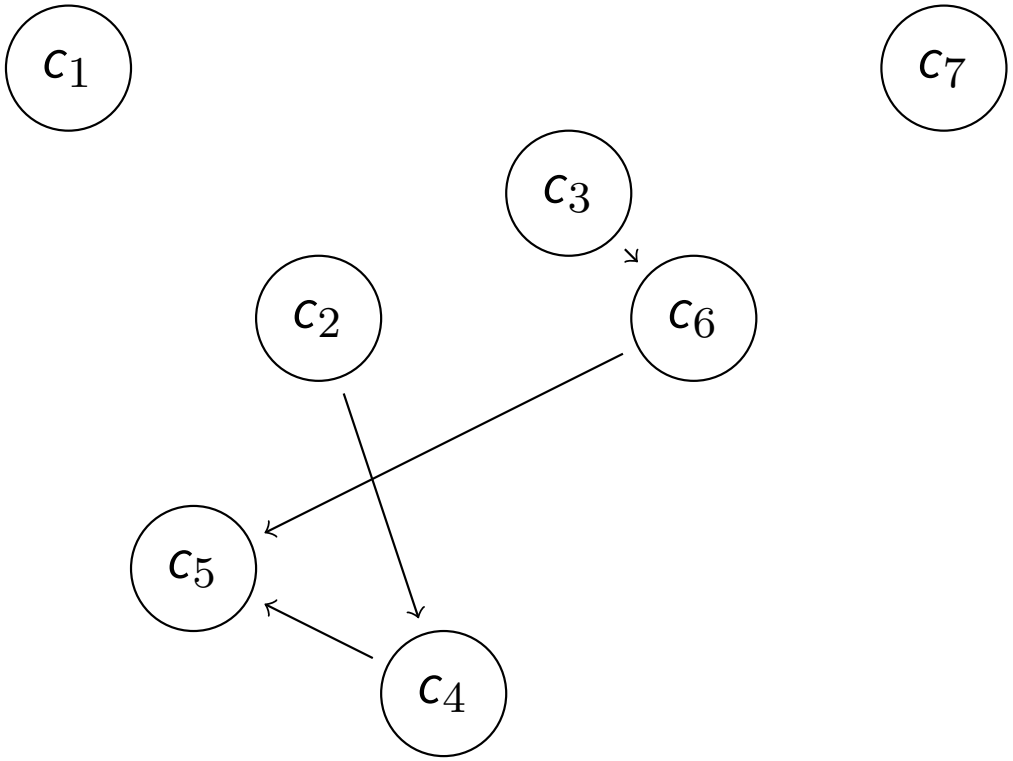



- connect 6 and 5

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| -1 | 4 | 6 | -2 | -3 | 5 | -1 |

- next connect 2 and 3

# Example, pt 5

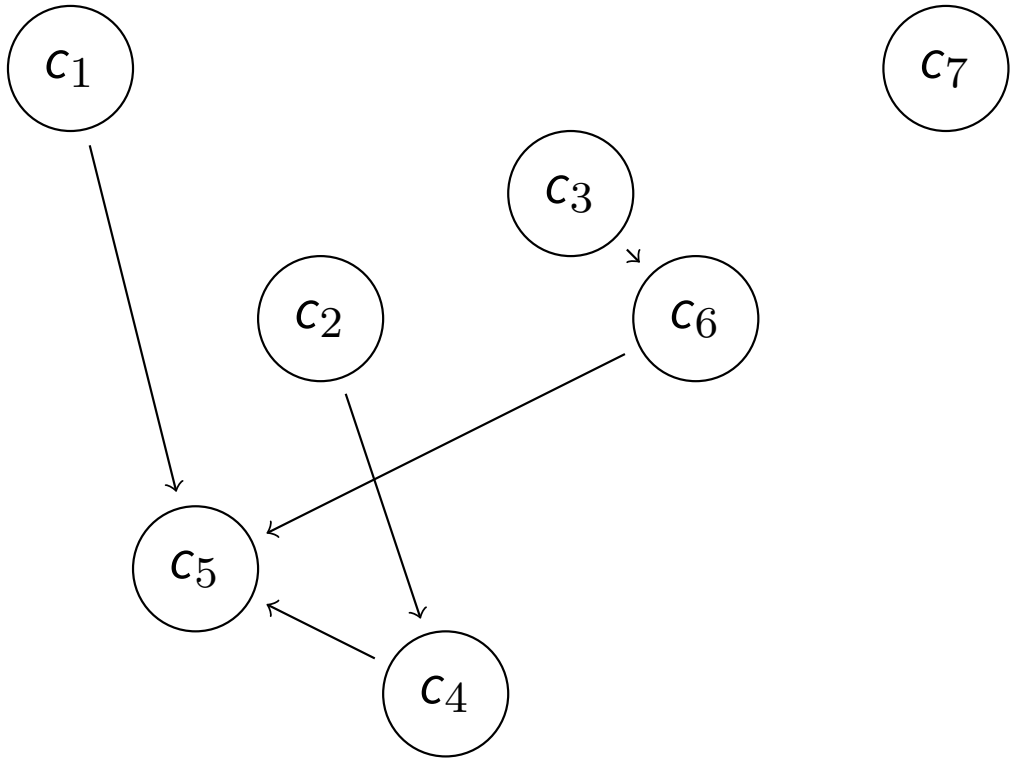



- connect 2 and 3 (will connect 4 to 5)

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| -1 | 4 | 6 | 5 | -5 | 5 | -1 |

- next connect 3 and 1

# Example, pt 6

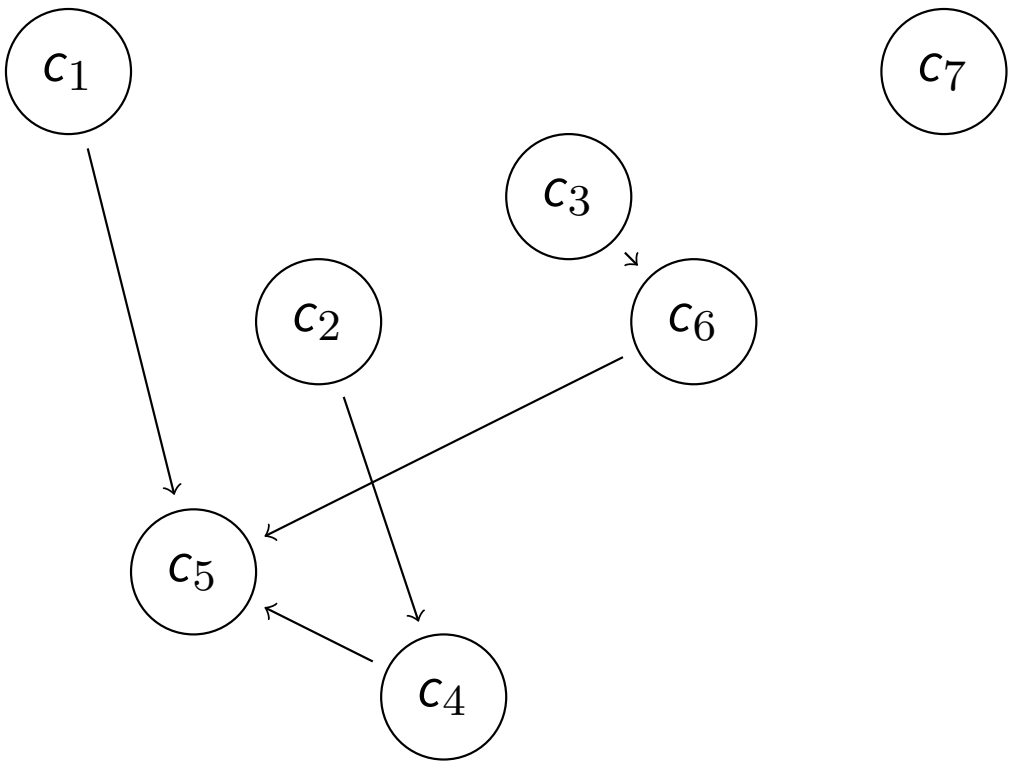



- connect 3 and 1

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| 5 | 4 | 6 | 5 | -6 | 5 | -1 |

# Path Compression

- One problem is that these paths can get quite long if we are unlucky.
- Solution: update `find` so it compresses the paths.

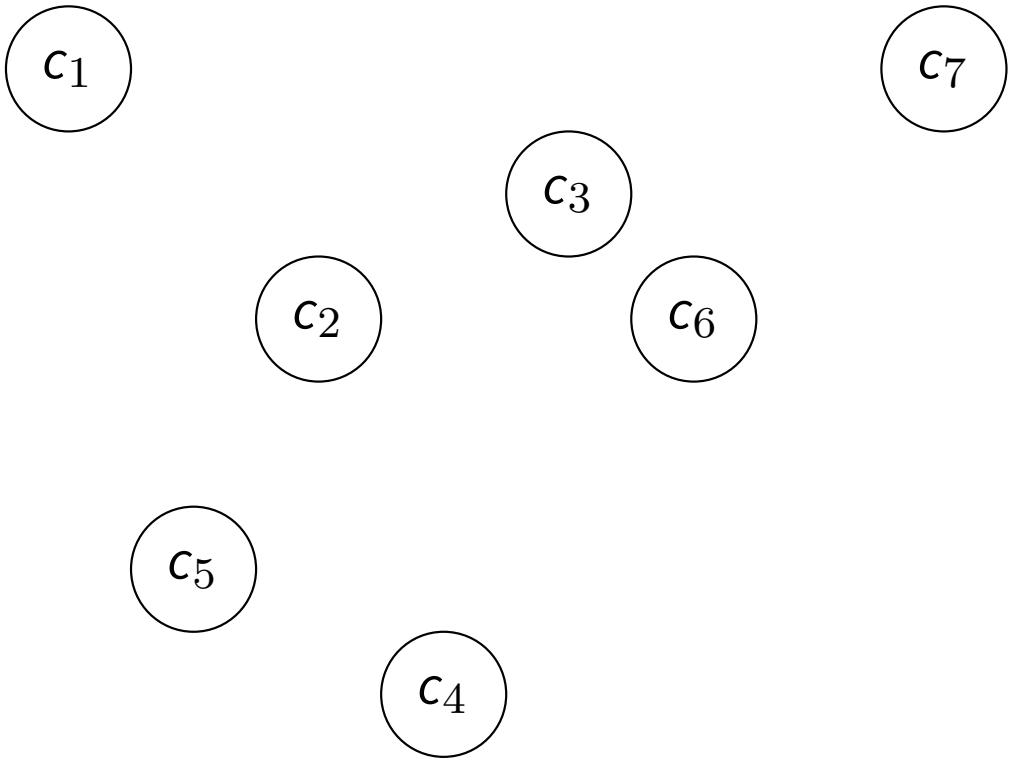


# Find with Path Compression

```
int find(int c, vi ds) {
  if (ds[c] < 0)
    return c;
  else
    return ds[c] = find(ds[c],ds);
}

int add(int c1, int c2, vi ds) {
  int sa = find(c1);
  int sb = find(c2);
  if (sa == sb)
    return sa;
  if (ds[sa] > ds[sb]) {
    ds[sb] += ds[sa]; ds[sa] = sb; return sb;
  } else {
    ds[sa] += ds[sb]; ds[sb] = sa; return sa;
  }
```

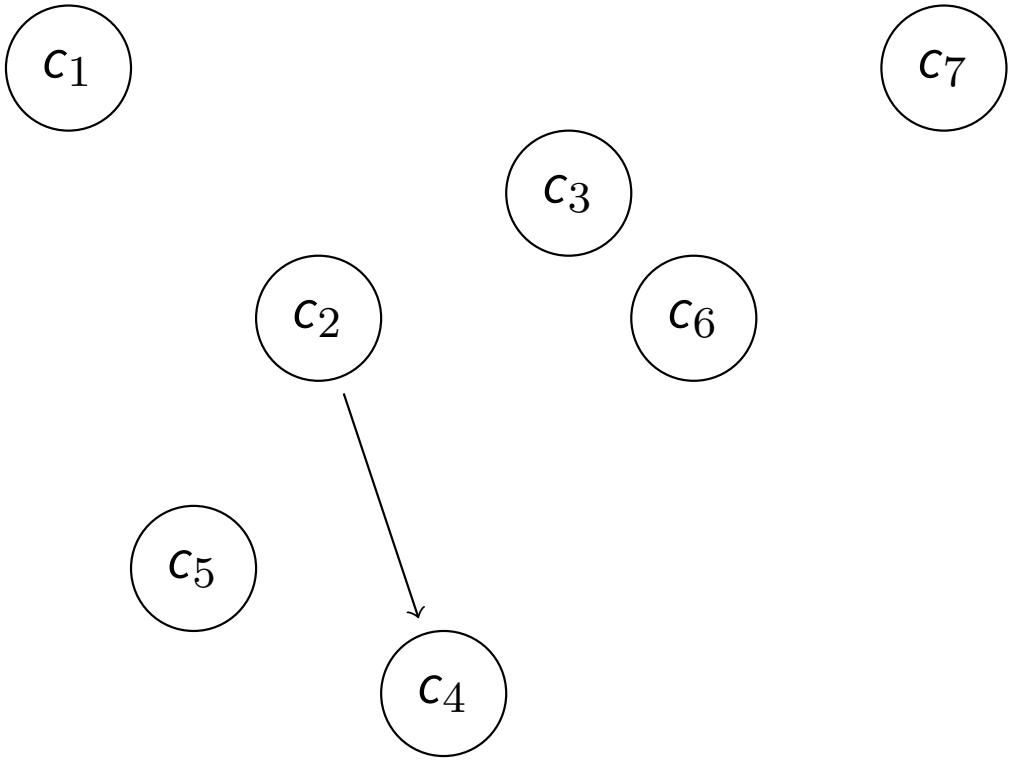
# Example, pt 1




| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| -1 | -1 | -1 | -1 | -1 | -1 | -1 |

- next connect 2 and 4

# Example, pt 2

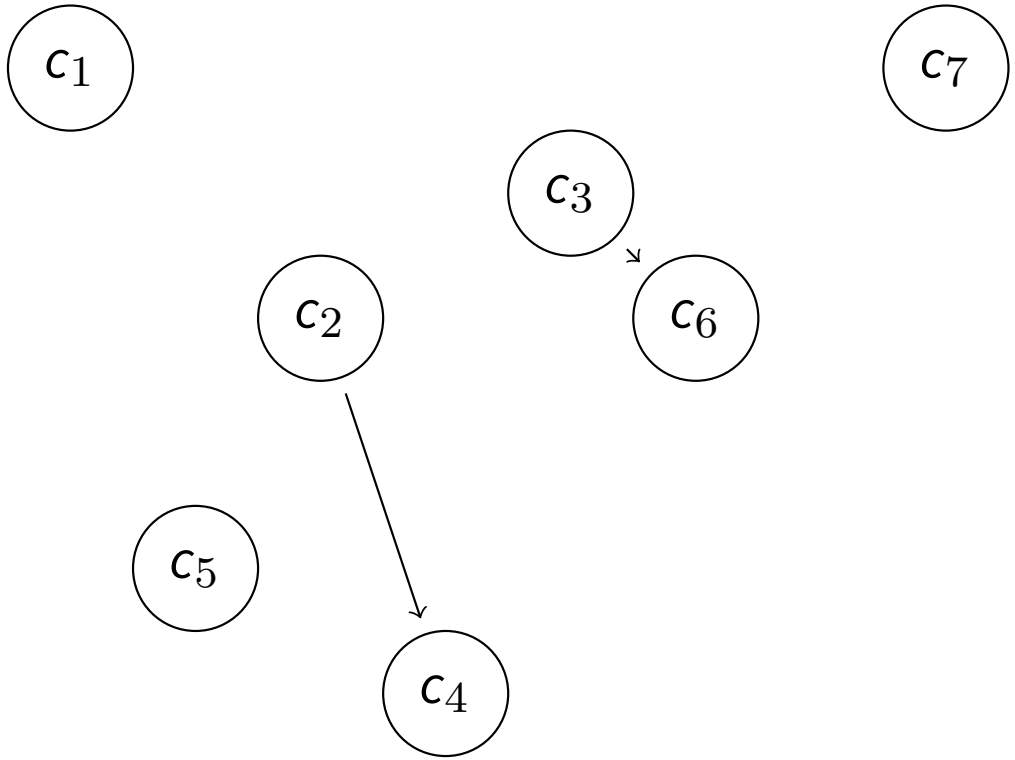



- connect 2 and 4

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| -1 | 4 | -1 | -2 | -1 | -1 | -1 |

- next connect 3 and 6

# Example, pt 3

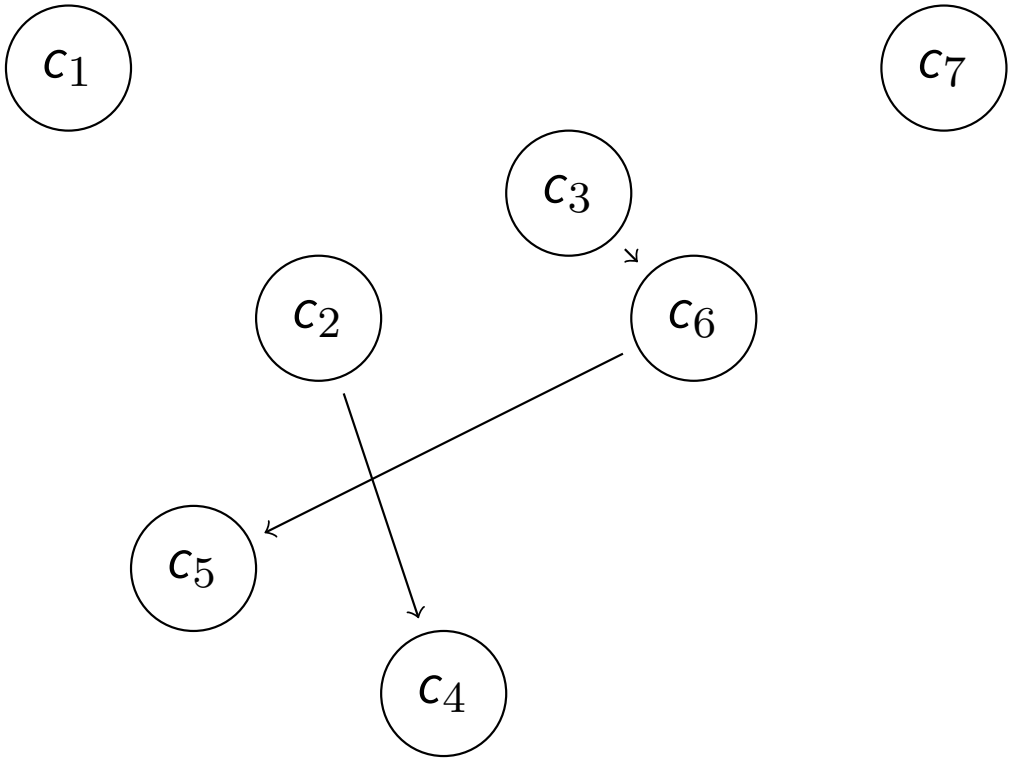



- connect 3 and 6

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| -1 | 4 | 6 | -2 | -1 | -2 | -1 |

- next connect 6 and 5

# Example, pt 4

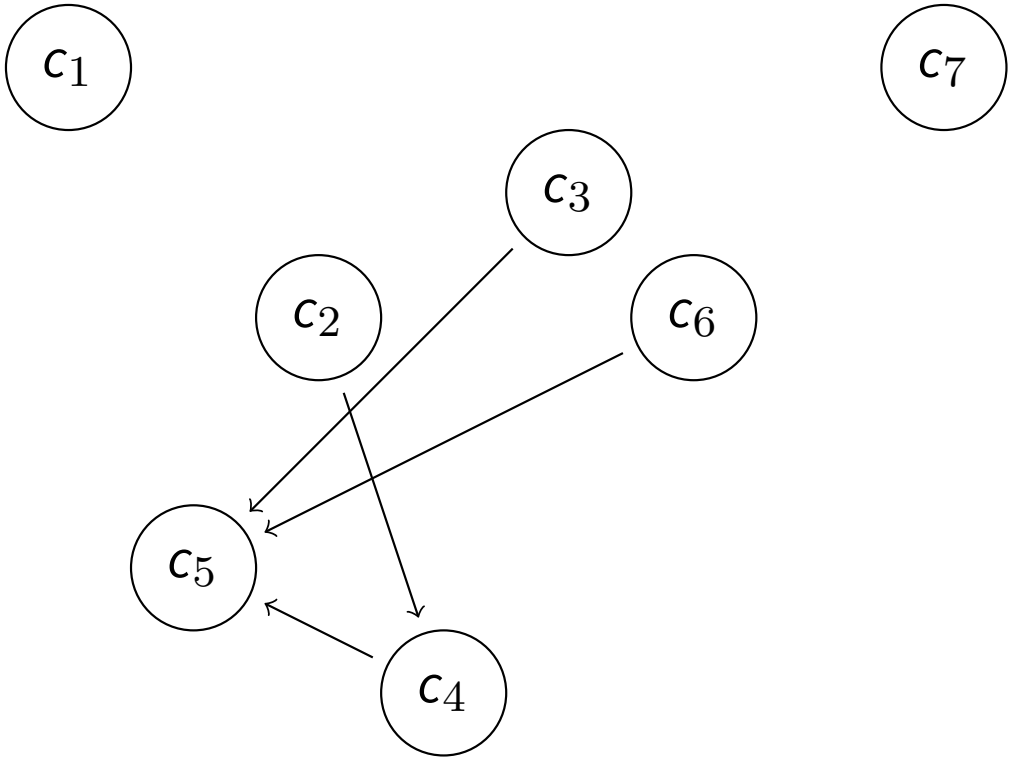



- connect 6 and 5

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| -1 | 4 | 6 | -2 | -3 | 5 | -1 |

- next connect 2 and 3

# Example, pt 5

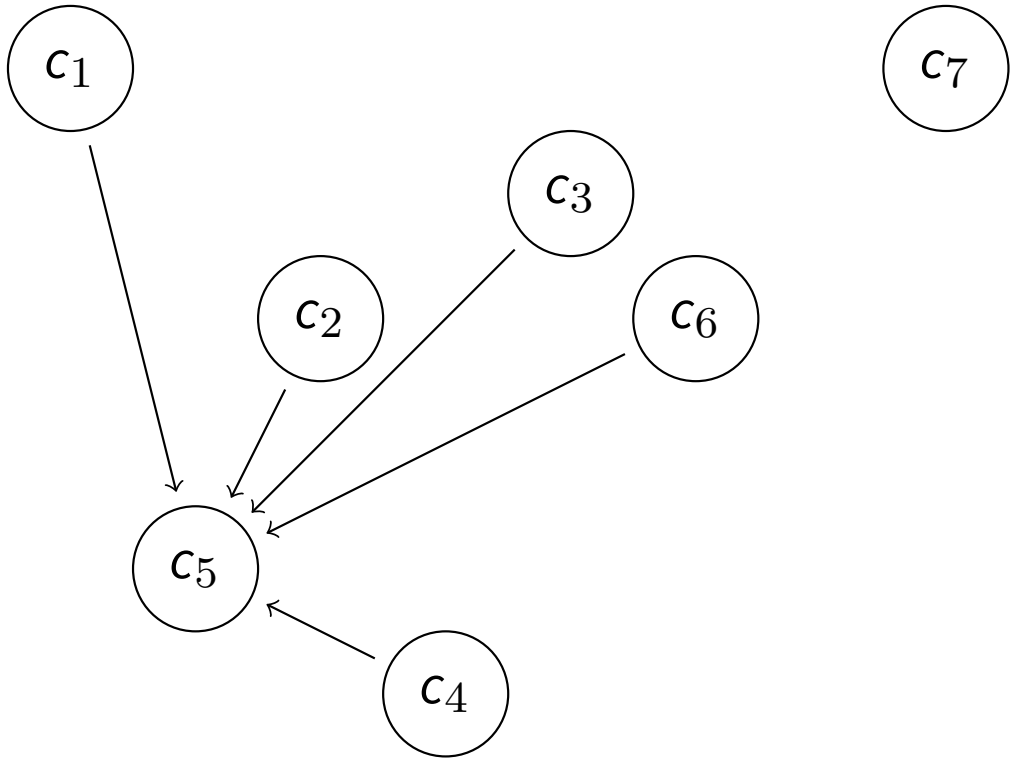



- connect 2 and 3 (will connect 4 to 5, and update 3)

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| -1 | 4 | 5 | 5 | -5 | 5 | -1 |

- next connect 3 and 1

# Example, pt 6




- connect 3 and 1

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| 5 | 5 | 5 | 5 | -6 | 5 | -1 |