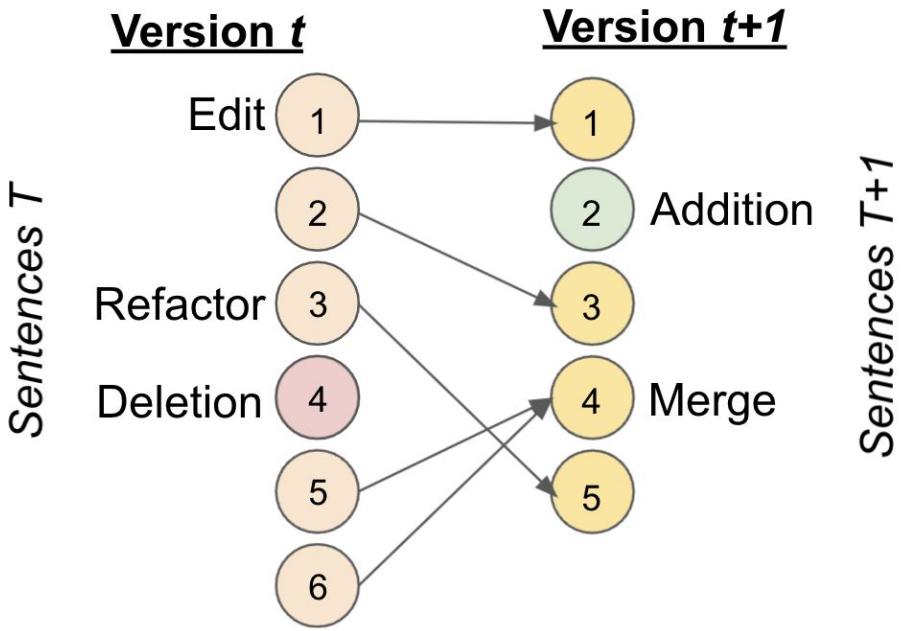
Refactoring in More Detail

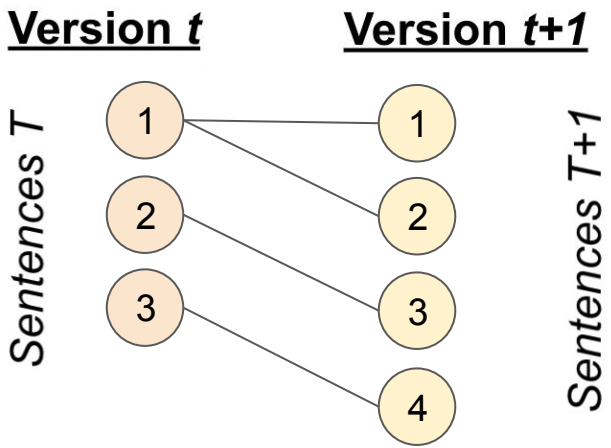
Alexander Spangher, USC
Carol Hu, UCLA

Purpose

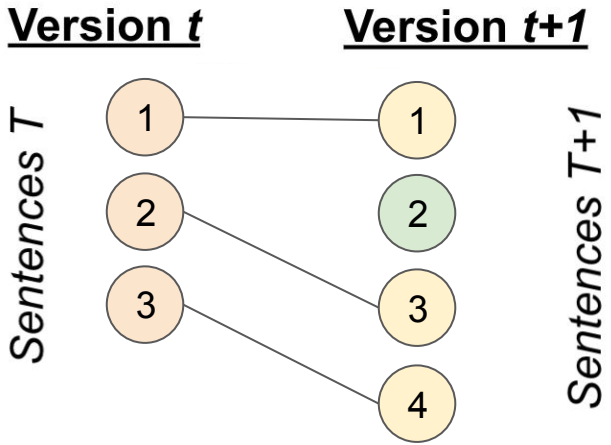
A refactor, in the context of *NewsEdits*, is a sentence that has been purposefully moved in the document. I.e. the editor made the conscious decision to move a sentence higher or lower.



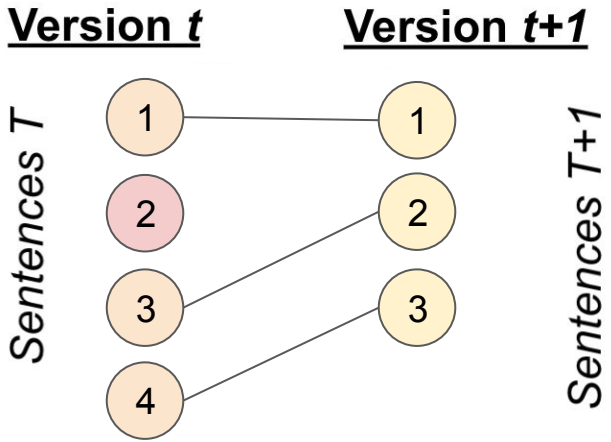
What is *not* a refactor? Sentence shifts that result from merges, splits, additions or deletions.



What is *not* a refactor? Sentence shifts that result from merges, splits, additions or deletions.

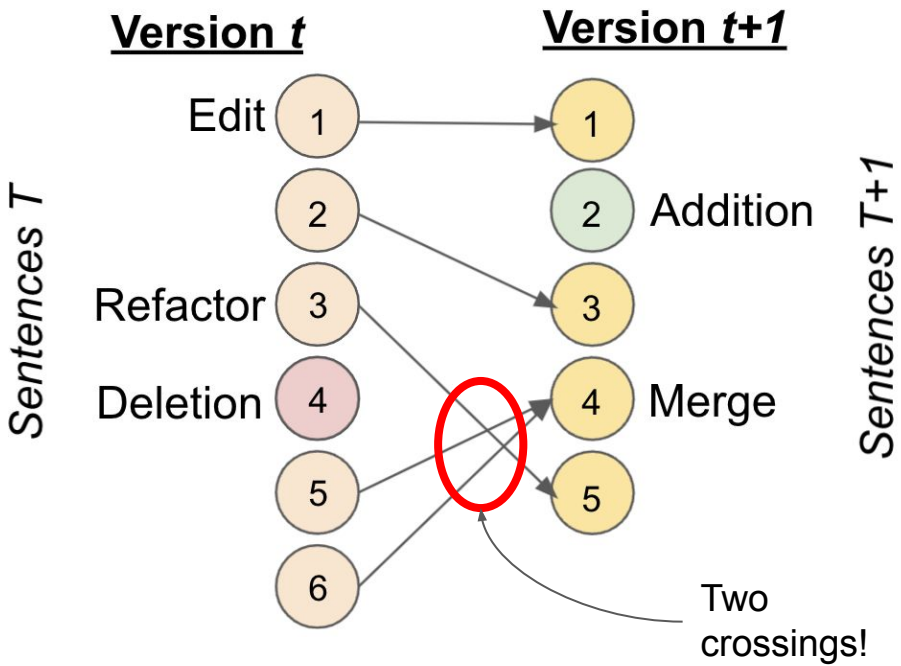


What is *not* a refactor? Sentence shifts that result from merges, splits, additions or deletions.



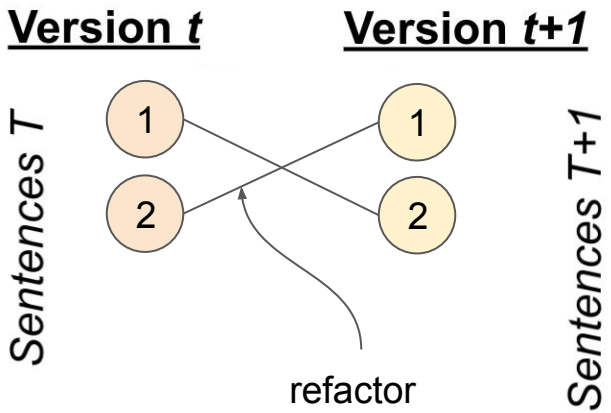
So, what is a refactor?

Heuristic #1: The nodes with edge(s) that have the most crossings.



So, what is a refactor?

Heuristic #2: If two edges have equal crossings, the one that moves upwards is the refactor.



So…

Given a dictionary of edge crossings, which we calculate separately:

Iterate through, labeling edge-crossings as refactors, and then remove that edge from the bipartite graph.

Continue until you have none left.

input : Sentence matches, i.e. edges e between doc i and doc j, as a list of tuples:
$e_i = (s_{i1}, s_{i2}), e_j = (s_{j1}, s_{j2}) \ldots$.
output : Minimal set of edges r that, when removed, eliminate all crossings.
// Subroutine identifies all edge crossings in e' and returns mapping $c = \{e_i \rightarrow [e_j, e_k \ldots], e_j \rightarrow \ldots\}$ from each edge to all its crossings.
$c = getEdgeCrossings(e)$
while $|c| > 0$ **do**
  // Find candidate set: all edges with maximum crossings.
  $m = \max_i |c[e'_i]|$
  $e' = e'_i$ where $|c[e'_i]| = m$
  if $|e'| > 1$ **then**
    // Filter candidate set: all edges $\in e'$ that extend the maximum distance.
    $d = \max_i |e'_i[0] - e'_i[1]|$
    $e' = e'_i$ where $|e'_i[0] - e'_i[1]| = d$
    if $|e'| > 1$ **then**
      // Filter candidate set: all edges $\in e'$ that move up.
      $e' = e'_i$ where $e'_i[1] - e'_i[0] < 0$
    else
  else
end
// Take first element of e' as the candidate to remove.
$t = e'[0]$
$r.push(t)$
// Remove t from c and from all $c[e'_i]$ lists that contain it.
$c = removeEdge(t)$

Algorithm 2: Identifying Refactors. We define refactors as the minimal set of edge crossings in a bipartite graph which, when removed, remove all edge crossings.

Tricky Edge-case Examples

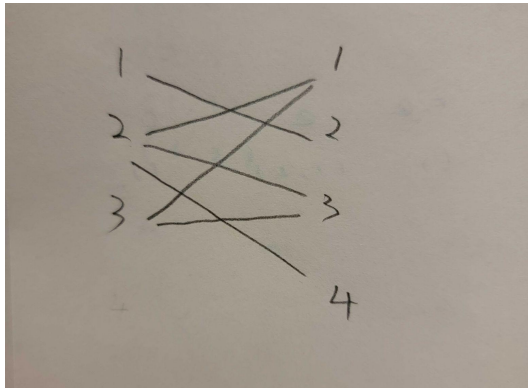
Template for reading examples

```
n = 3   # nodes in version t
m = 4   # nodes in version t+1
k = 6   # edges
e = [   # node index of edge crossings
    (1, 2),
    (2, 1),
    (3, 1),
    (2, 3),
    (3, 3),
    (2, 4)
]
```

Dictionary of edge-crossings at step 1

```
{(1, 2): {(2, 1), (3, 1)},
 (2, 1): {(1, 2)},
 (2, 3): {(3, 1)},
 (2, 4): {(3, 1), (3, 3)},
 (3, 1): {(1, 2), (2, 3), (2, 4)},
 (3, 3): {(2, 4)}}
```

Visualized diagram



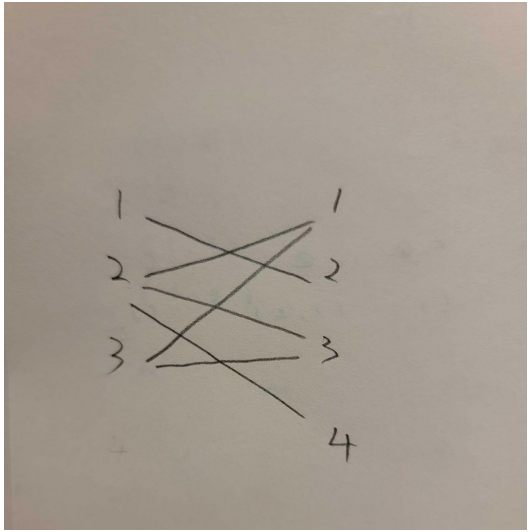

```
remove :  (3, 1)
(1, 2)  :  {(2, 1)}
(2, 1)  :  {(1, 2)}
(2, 3)  :  set()
(2, 4)  :  {(3, 3)}
(3, 3)  :  {(2, 4)}

remove :  (2, 1)
(1, 2)  :  set()
(2, 3)  :  set()
(2, 4)  :  {(3, 3)}
(3, 3)  :  {(2, 4)}

remove :  (2, 4)
(1, 2)  :  set()
(2, 3)  :  set()
(3, 3)  :  set()
```

Iterations of edge removals and resulting edge-crossing dictionaries

Example #1

```
n = 3
m = 4
k = 6
e = [
    (1, 2),
    (2, 1),
    (3, 1),
    (2, 3),
    (3, 3),
    (2, 4)
]
```

```
{(1, 2): {(2, 1), (3, 1)},
 (2, 1): {(1, 2)},
 (2, 3): {(3, 1)},
 (2, 4): {(3, 1), (3, 3)},
 (3, 1): {(1, 2), (2, 3), (2, 4)},
 (3, 3): {(2, 4)}}
```

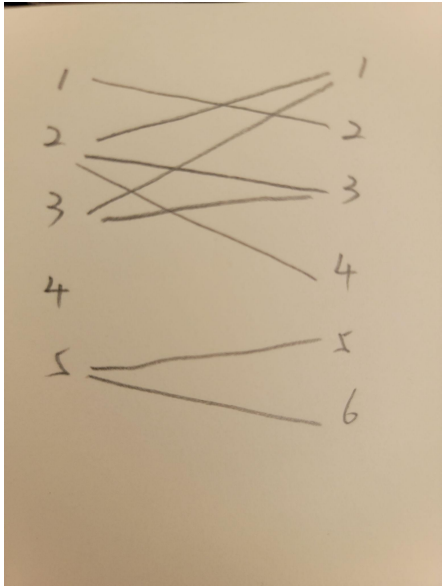



```
remove :  (3, 1)
(1, 2)  :  {(2, 1)}
(2, 1)  :  {(1, 2)}
(2, 3)  :  set()
(2, 4)  :  {(3, 3)}
(3, 3)  :  {(2, 4)}

remove :  (2, 1)
(1, 2)  :  set()
(2, 3)  :  set()
(2, 4)  :  {(3, 3)}
(3, 3)  :  {(2, 4)}

remove :  (2, 4)
(1, 2)  :  set()
(2, 3)  :  set()
(3, 3)  :  set()
```

Example #2

```
n = 5
m = 6
k = 8
e = [
    (1, 2),
    (2, 1),
    (3, 1),
    (2, 3),
    (3, 3),
    (2, 4),
    (5, 5),
    (5, 6)
]
```

```
{(1, 2): {(2, 1), (3, 1)},
 (2, 1): {(1, 2)},
 (2, 3): {(3, 1)},
 (2, 4): {(3, 1), (3, 3)},
 (3, 1): {(1, 2), (2, 3), (2, 4)},
 (3, 3): {(2, 4)},
 (5, 5): set(),
 (5, 6): set()}
```




```
remove :  (3, 1)
(1, 2)  :  {(2, 1)}
(2, 1)  :  {(1, 2)}
(2, 3)  :  set()
(2, 4)  :  {(3, 3)}
(3, 3)  :  {(2, 4)}
(5, 5)  :  set()
(5, 6)  :  set()
```

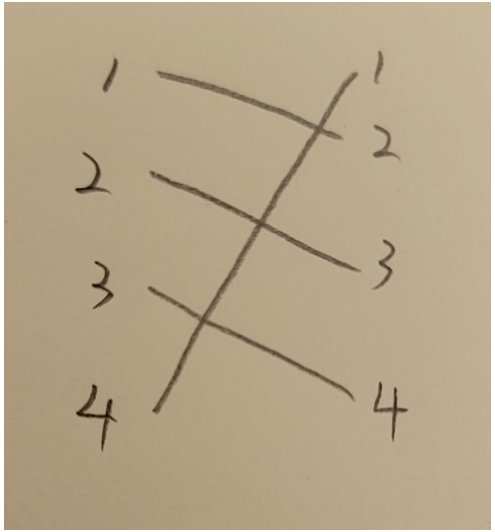
```
remove :  (2, 1)
(1, 2)  :  set()
(2, 3)  :  set()
(2, 4)  :  {(3, 3)}
(3, 3)  :  {(2, 4)}
(5, 5)  :  set()
(5, 6)  :  set()
```

```
remove :  (2, 4)
(1, 2)  :  set()
(2, 3)  :  set()
(3, 3)  :  set()
(5, 5)  :  set()
(5, 6)  :  set()
```

Example #3

```
n = 4
m = 4
k = 4
e = [
    (1, 2),
    (2, 3),
    (3, 4),
    (4, 1)
]
```

```
{(1, 2): {(4, 1)},
 (2, 3): {(4, 1)},
 (3, 4): {(4, 1)},
 (4, 1): {(1, 2), (2, 3), (3, 4)}}
```

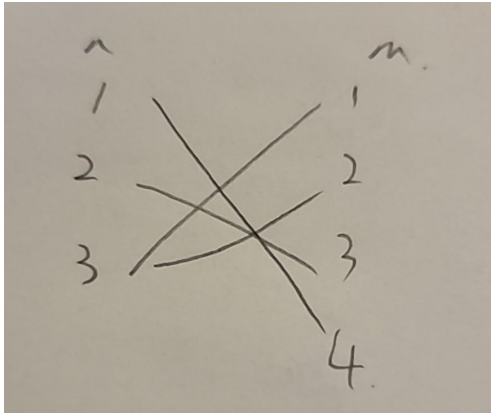



```
remove :  (4, 1)
(1, 2)  :  set()
(2, 3)  :  set()
(3, 4)  :  set()
```

Example #4

```
n = 3
m = 4
k = 4
e = [
    (1, 4),
    (2, 3),
    (3, 1),
    (3, 2)
]
```

```
{(1, 4): {(2, 3), (3, 1), (3, 2)},
 (2, 3): {(1, 4), (3, 1), (3, 2)},
 (3, 1): {(1, 4), (2, 3)},
 (3, 2): {(1, 4), (2, 3)}}
```

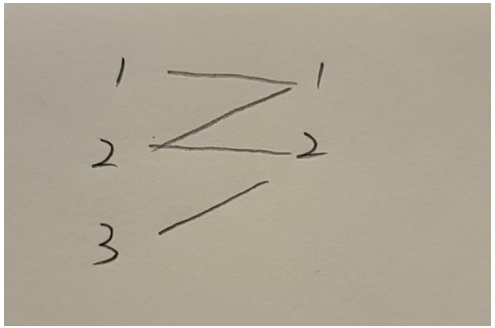



```
remove :  (1, 4)
(2, 3)  :  {(3, 1), (3, 2)}
(3, 1)  :  {(2, 3)}
(3, 2)  :  {(2, 3)}

remove :  (2, 3)
(3, 1)  :  set()
(3, 2)  :  set()
```

Example #5

```
{(1, 1): set(), (2, 1): set(), (2, 2): set(), (3, 2): set()}
```

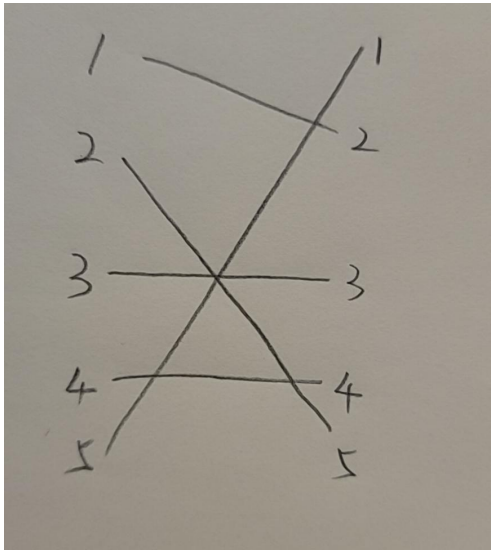
```
n = 3
m = 2
k = 4
e = [
    (1, 1),
    (2, 1),
    (3, 2),
    (2, 2)
]
```



Example #6

```
n = 5
m = 5
k = 5
e = [
    (1, 2),
    (5, 1),
    (3, 3),
    (4, 4),
    (2, 5)
]
```

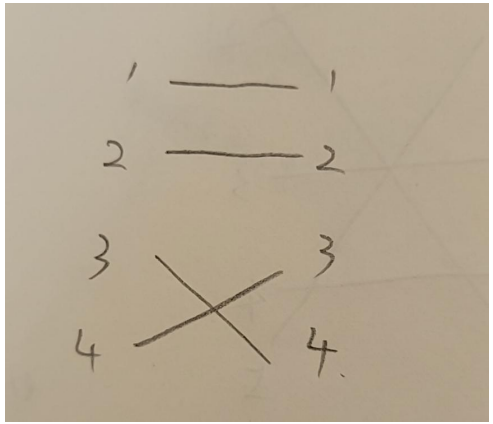
```
{(1, 2): {(5, 1)},
 (2, 5): {(3, 3), (4, 4), (5, 1)},
 (3, 3): {(2, 5), (5, 1)},
 (4, 4): {(2, 5), (5, 1)},
 (5, 1): {(1, 2), (2, 5), (3, 3), (4, 4)}}
```




```
remove :  (5, 1)
(1, 2)  :  set()
(2, 5)  :  {(4, 4), (3, 3)}
(3, 3)  :  {(2, 5)}
(4, 4)  :  {(2, 5)}

remove :  (2, 5)
(1, 2)  :  set()
(3, 3)  :  set()
(4, 4)  :  set()
```

Example #7

```
n = 4
m = 4
k = 4
e = [
    (1, 1),
    (1, 2),
    (3, 4),
    (4, 3),
]
```

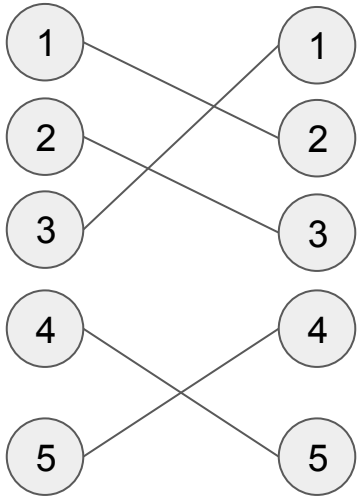



```
(1, 1)  :  set()
(1, 2)  :  set()
(3, 4)  :  {(4, 3)}
(4, 3)  :  {(3, 4)}
```

```
remove :  (4, 3)
(1, 1)  :  set()
(1, 2)  :  set()
(3, 4)  :  set()
```

Example #8

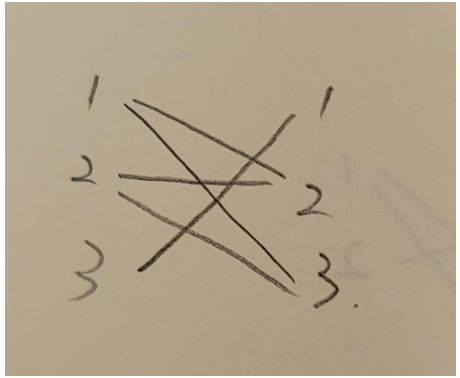
```
n = 5
m = 5
k = 5
e = [
    (1, 2),
    (2, 3),
    (3, 1),
    (4, 5),
    (5, 4)
]
```



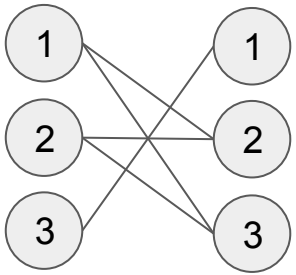

```
(1, 2)  :  {(3, 1)}
(2, 3)  :  {(3, 1)}
(3, 1)  :  {(2, 3), (1, 2)}
(4, 5)  :  {(5, 4)}
(5, 4)  :  {(4, 5)}
```

```
remove :  (3, 1)
(1, 2)  :  set()
(2, 3)  :  set()
(4, 5)  :  {(5, 4)}
(5, 4)  :  {(4, 5)}

remove :  (5, 4)
(1, 2)  :  set()
(2, 3)  :  set()
(4, 5)  :  set()
```

Example #9 - Question



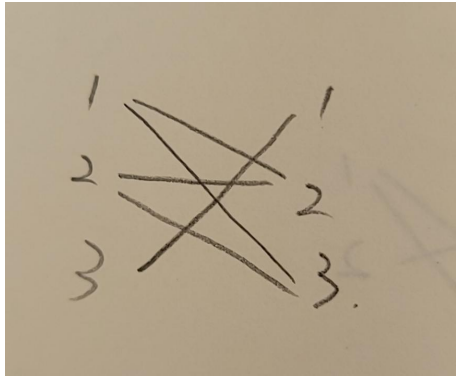

```
n = 3
m = 3
k = 5
e = [
    (1, 3),
    (3, 1),
    (1, 2),
    (2, 2),
    (2, 3)
]
```



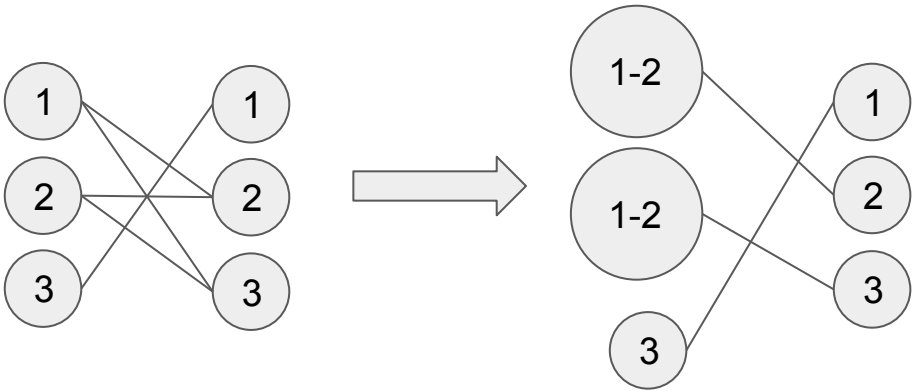

```
{(1, 2): {(3, 1)},
 (1, 3): {(2, 2), (3, 1)},
 (2, 2): {(1, 3), (3, 1)},
 (2, 3): {(3, 1)},
 (3, 1): {(1, 2), (1, 3), (2, 2), (2, 3)}}
```

```
remove :  (3, 1)
(1, 2)  :  set()
(1, 3)  :  {(2, 2)}
(2, 2)  :  {(1, 3)}
(2, 3)  :  set()

remove :  (1, 3)
(1, 2)  :  set()
(2, 2)  :  set()
(2, 3)  :  set()
```

Example #9 - Question




```
{(1, 2): {(3, 1)},
 (1, 3): {(2, 2), (3, 1)},
 (2, 2): {(1, 3), (3, 1)},
 (2, 3): {(3, 1)},
 (3, 1): {(1, 2), (1, 3), (2, 2), (2, 3)}}
```




```
remove :  (3, 1)
(1, 2)  :  set()
(1, 3)  :  {(2, 2)}
(2, 2)  :  {(1, 3)}
(2, 3)  :  set()

remove :  (1, 3)
(1, 2)  :  set()
(2, 2)  :  set()
(2, 3)  :  set()
```