Advanced Data Structures

William Fiset

Union Find Outline

- **Discussion & Examples**
  - What is Union Find?
  - Magnets example
  - When and where is a Union Find used?
    - Kruskal’s minimum spanning tree algorithm
    - Complexity analysis
- **Implementation Details**
  - Find & Union operations
  - Path compression
- **Code Implementation**

Fenwick Tree Outline

- **Discussion & Examples**
  - Data structure motivation
  - What is a Fenwick tree?
  - Complexity analysis
- **Implementation details**
  - Range query
  - Point Updates
  - Fenwick tree construction
- **Code Implementation**

Union Find!

(Disjoint Set)

Discussion and Examples

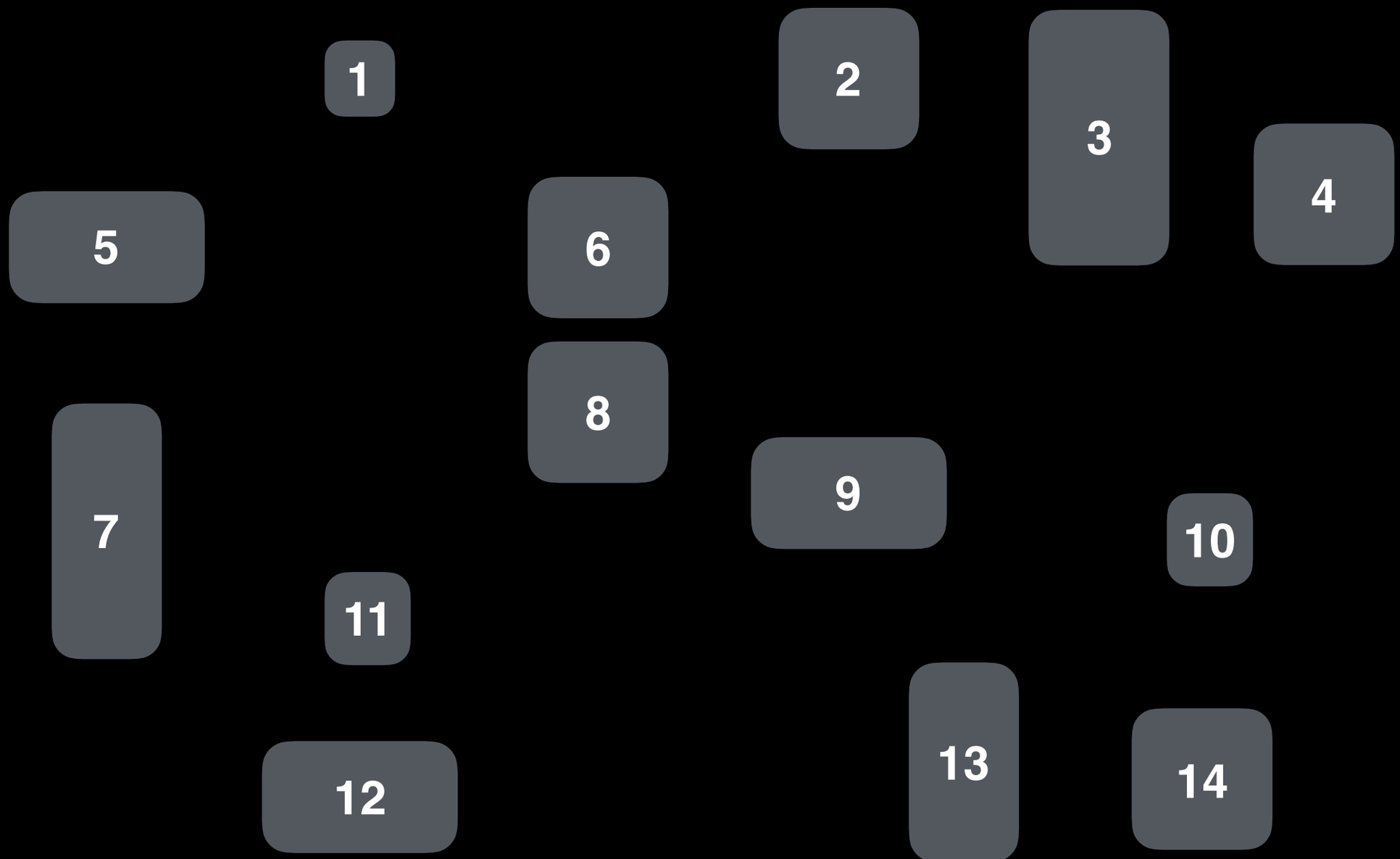
What is Union Find?

Union Find is a data structure that keeps track of elements which are split into one or more disjoint sets. Its has two primary operations: *find* and *union*.

Union Find Magnets Example

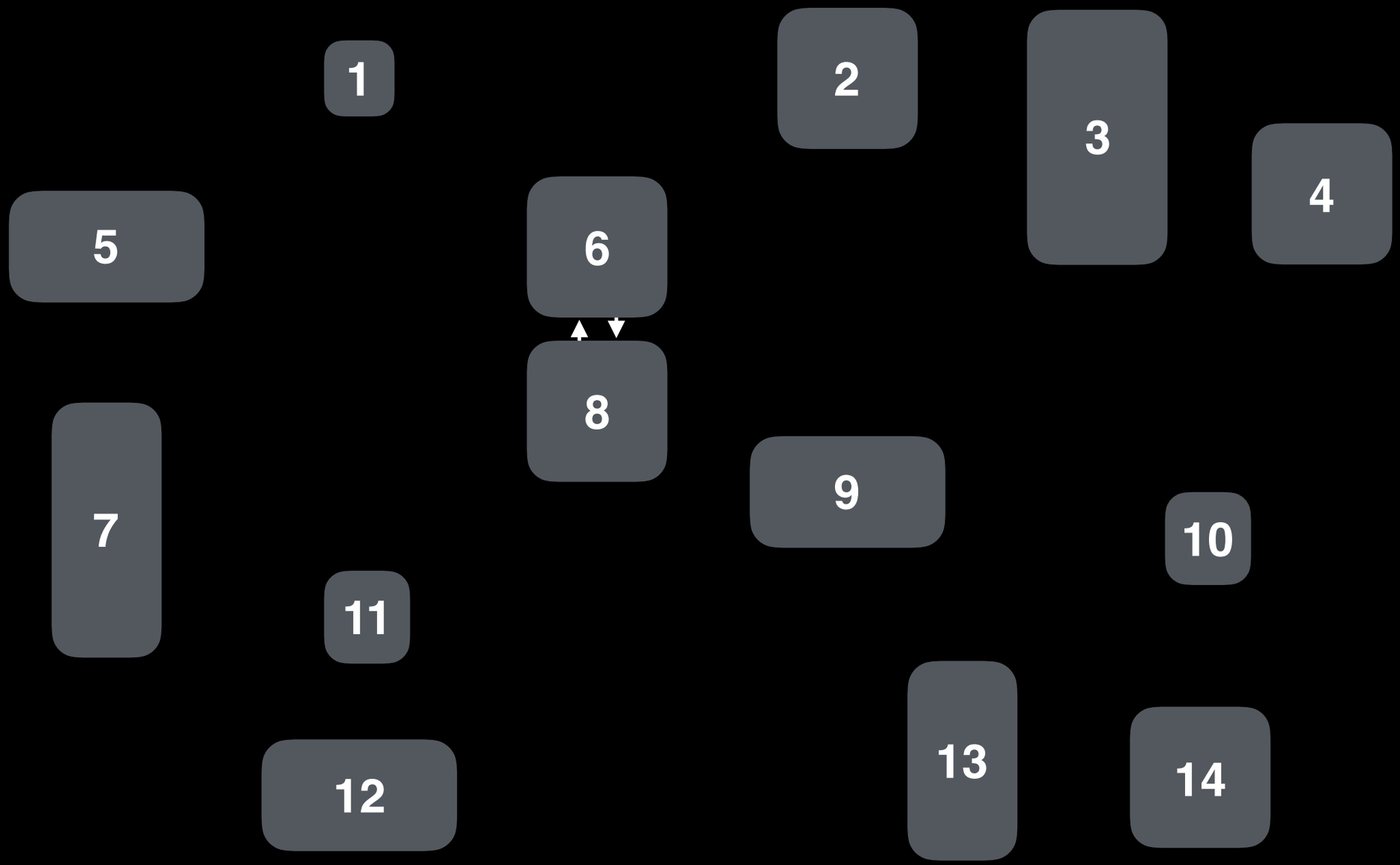
Union Find Magnets Example

Magnet 1
Magnet 2
Magnet 3
Magnet 4
Magnet 5
Magnet 6
Magnet 7
Magnet 8
Magnet 9
Magnet 10
Magnet 11
Magnet 12
Magnet 13
Magnet 14



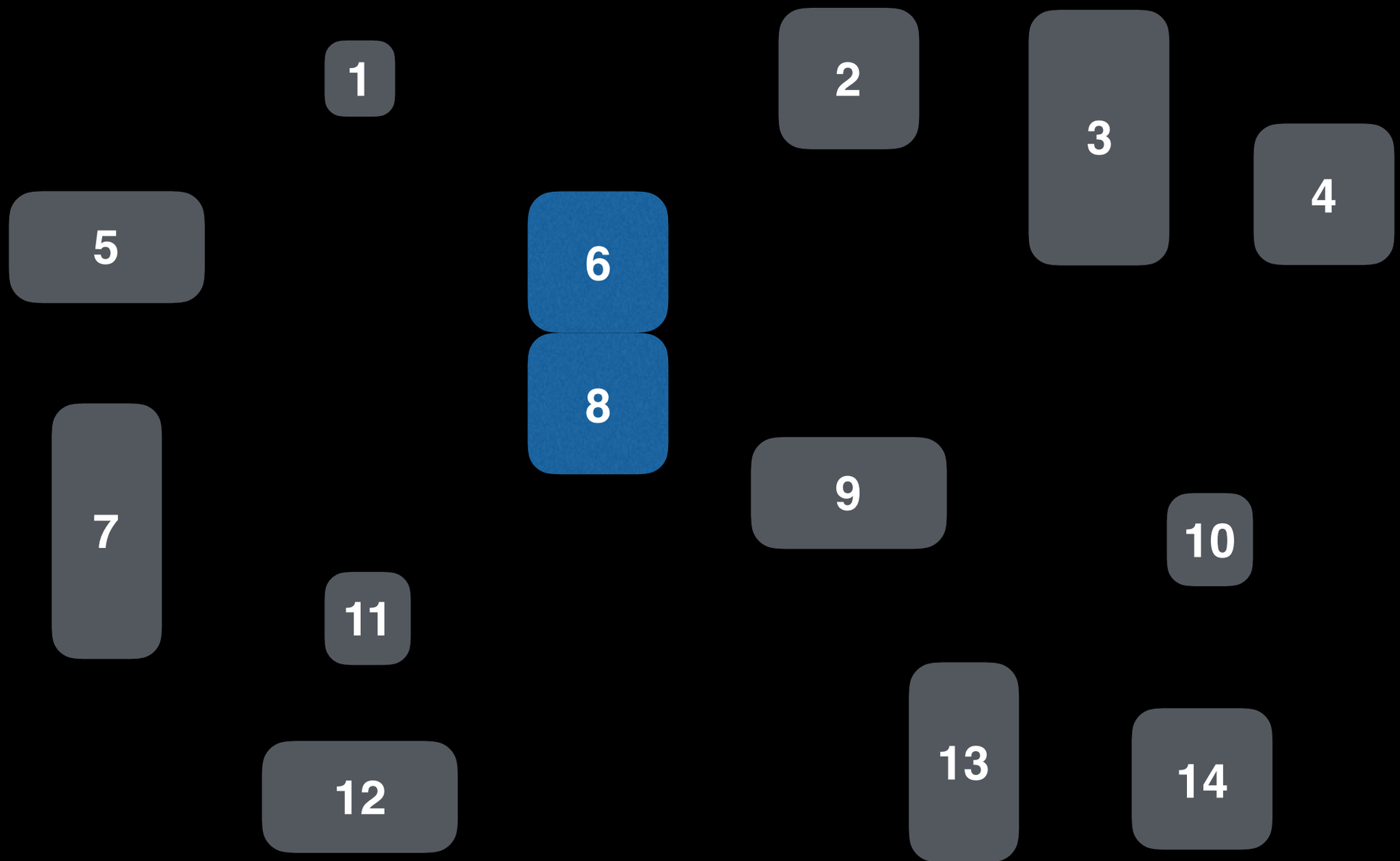
Union Find Magnets Example

Magnet 1
Magnet 2
Magnet 3
Magnet 4
Magnet 5
Magnet 6
Magnet 7
Magnet 8
Magnet 9
Magnet 10
Magnet 11
Magnet 12
Magnet 13
Magnet 14



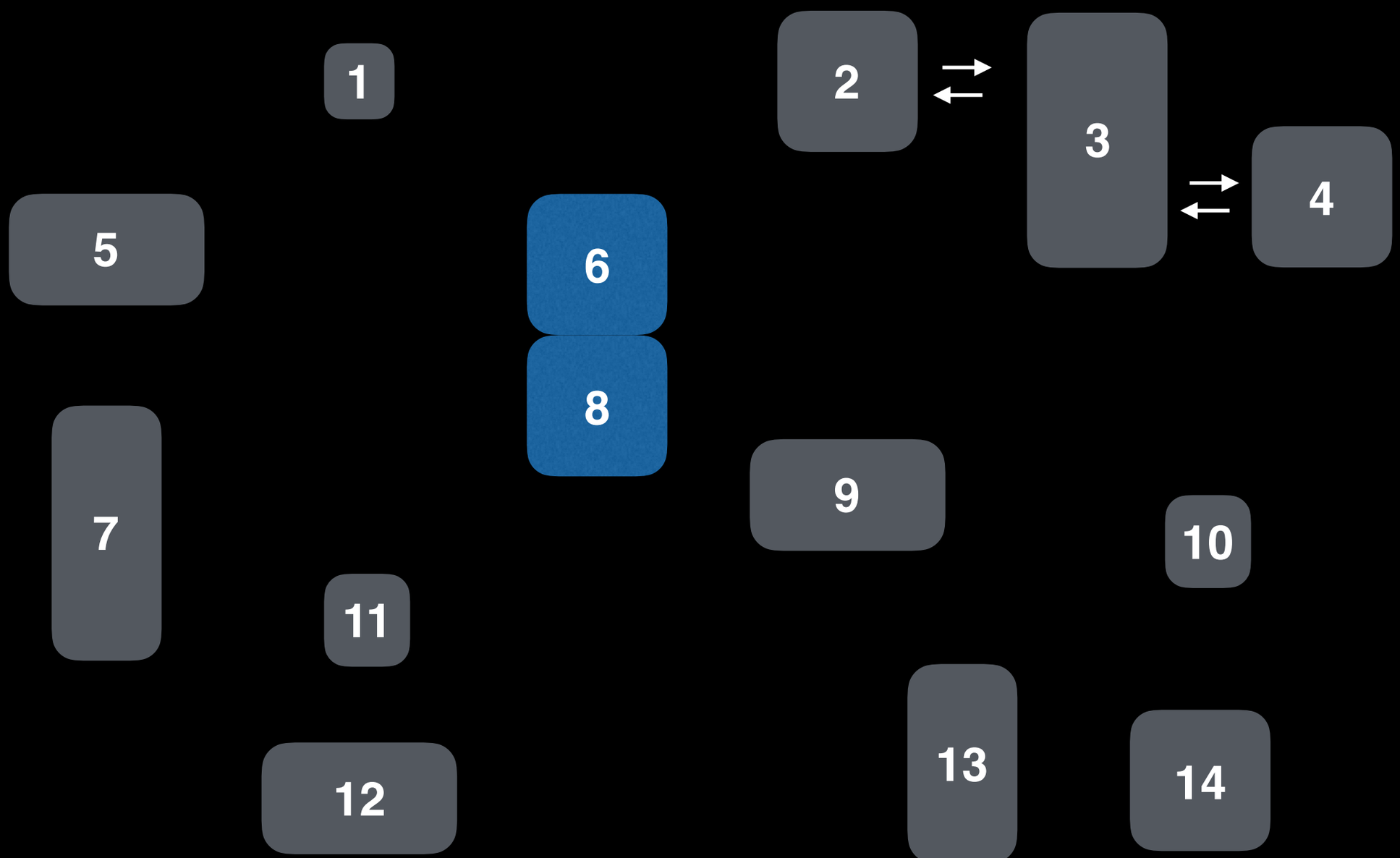
Union Find Magnets Example

Magnet 1
Magnet 2
Magnet 3
Magnet 4
Magnet 5
Magnet 6
Magnet 7
Magnet 8
Magnet 9
Magnet 10
Magnet 11
Magnet 12
Magnet 13
Magnet 14



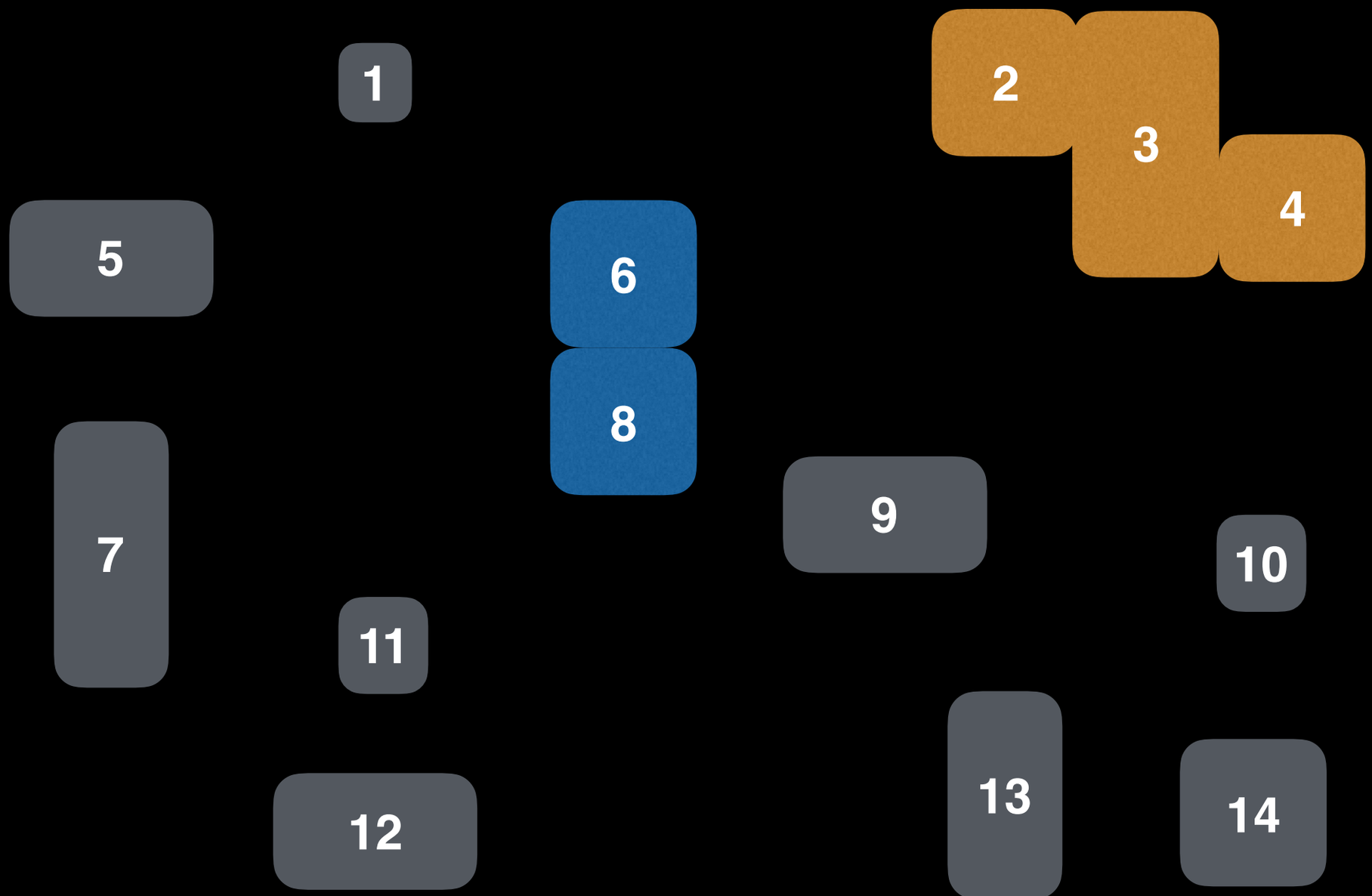
Union Find Magnets Example

Magnet 1
Magnet 2
Magnet 3
Magnet 4
Magnet 5
Magnet 6
Magnet 7
Magnet 8
Magnet 9
Magnet 10
Magnet 11
Magnet 12
Magnet 13
Magnet 14



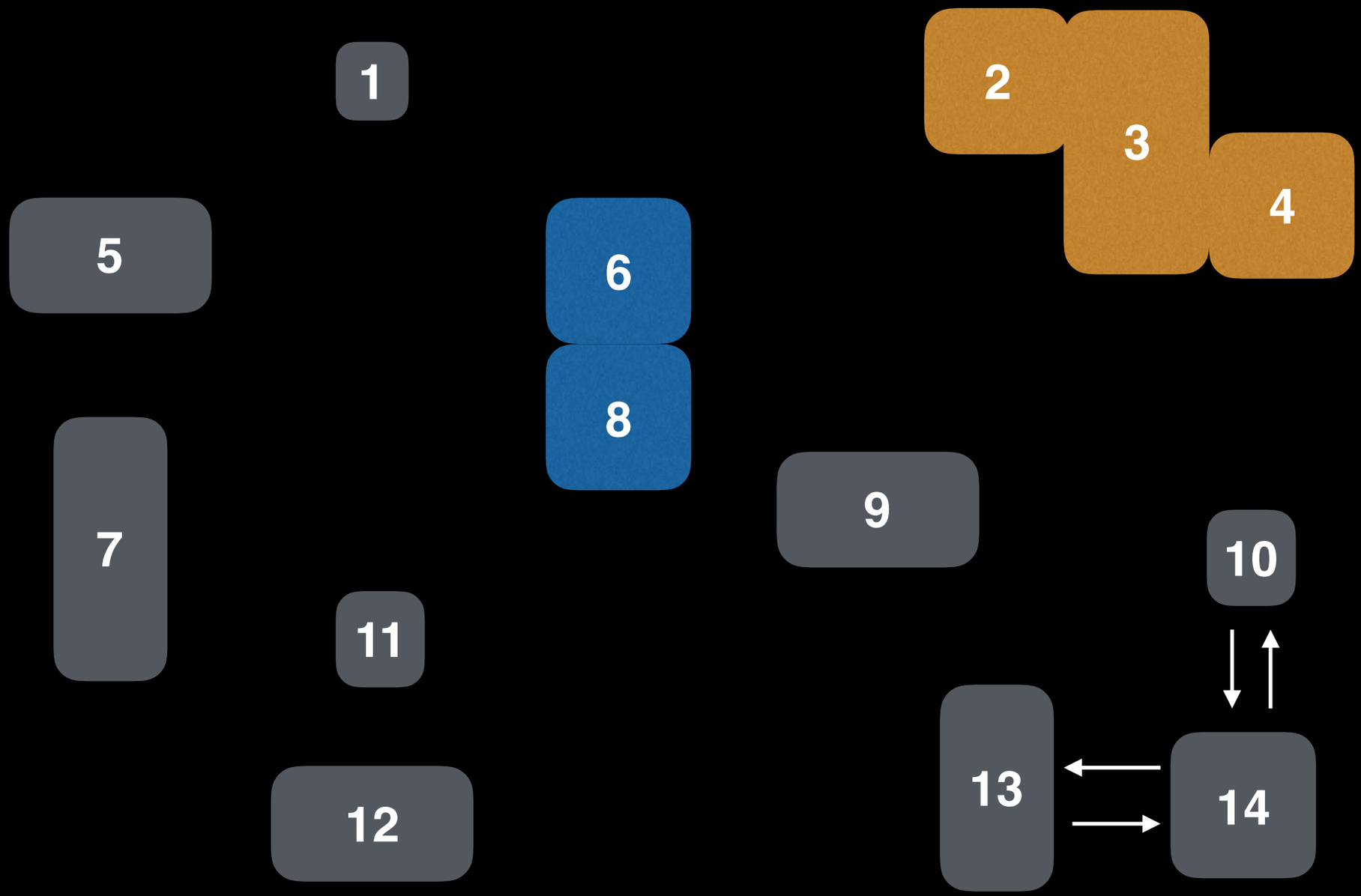
Union Find Magnets Example

Magnet 1
Magnet 2
Magnet 3
Magnet 4
Magnet 5
Magnet 6
Magnet 7
Magnet 8
Magnet 9
Magnet 10
Magnet 11
Magnet 12
Magnet 13
Magnet 14



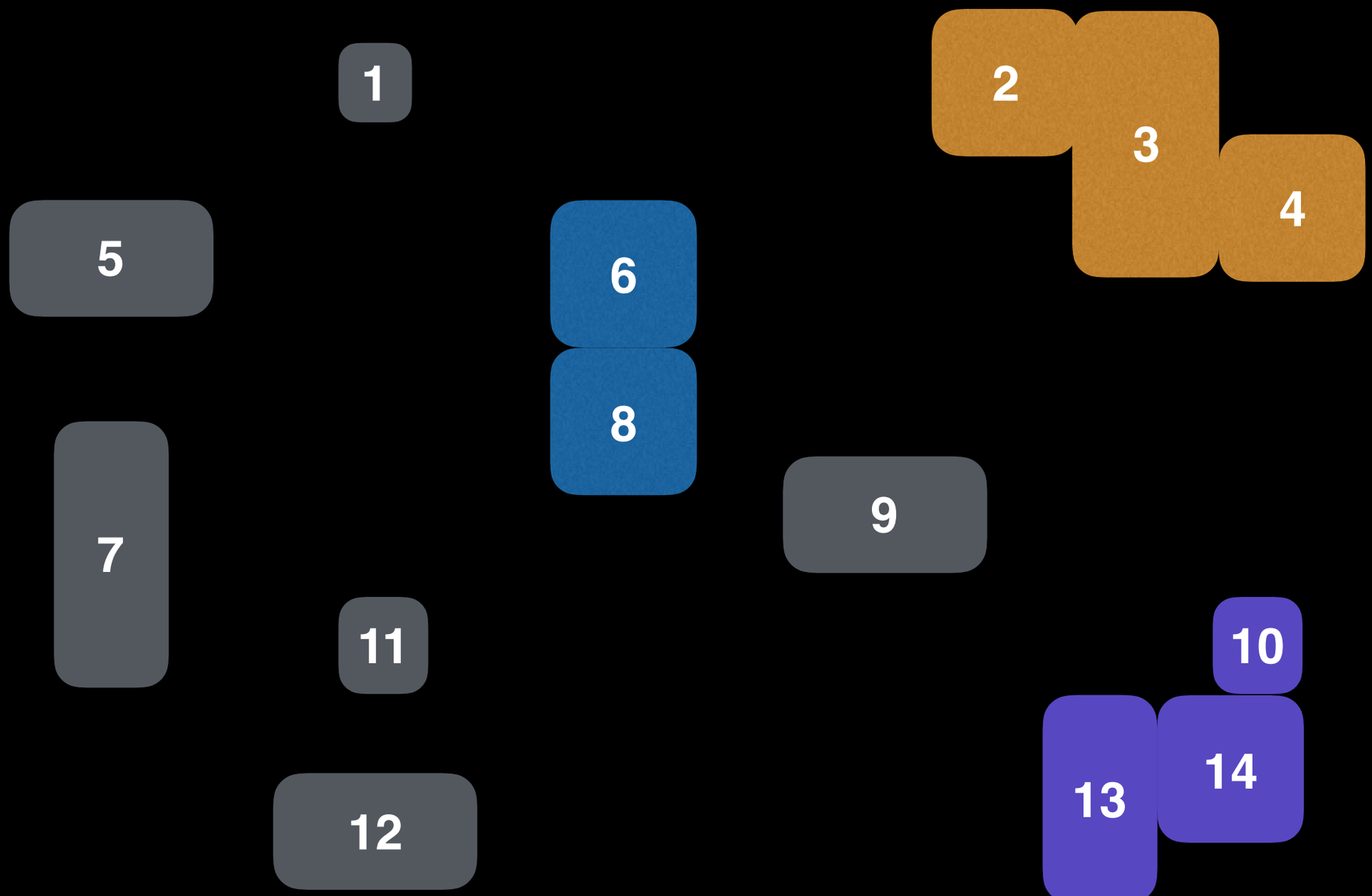
Union Find Magnets Example

Magnet 1
Magnet 2
Magnet 3
Magnet 4
Magnet 5
Magnet 6
Magnet 7
Magnet 8
Magnet 9
Magnet 10
Magnet 11
Magnet 12
Magnet 13
Magnet 14



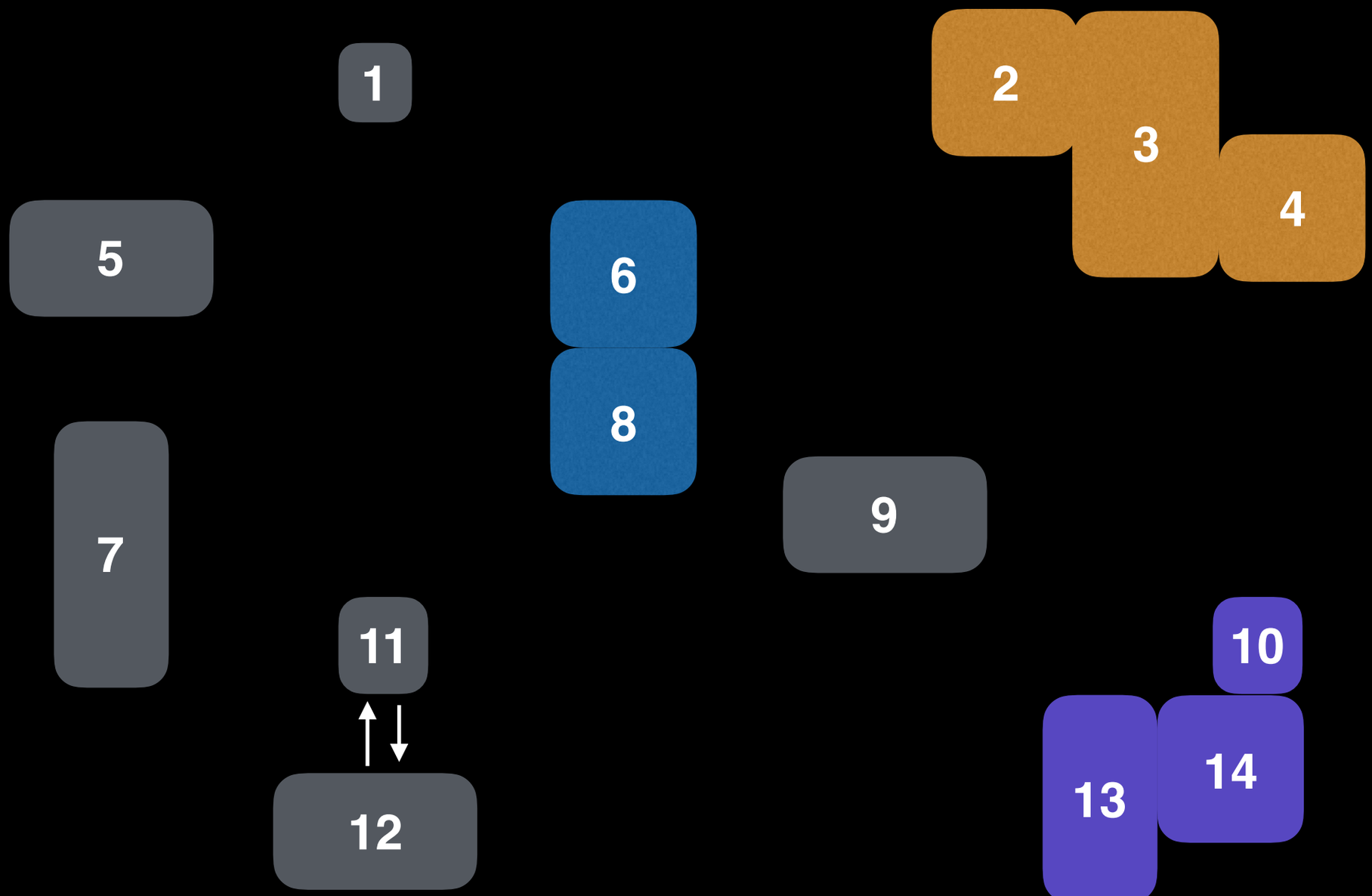
Union Find Magnets Example

Magnet 1
Magnet 2
Magnet 3
Magnet 4
Magnet 5
Magnet 6
Magnet 7
Magnet 8
Magnet 9
Magnet 10
Magnet 11
Magnet 12
Magnet 13
Magnet 14



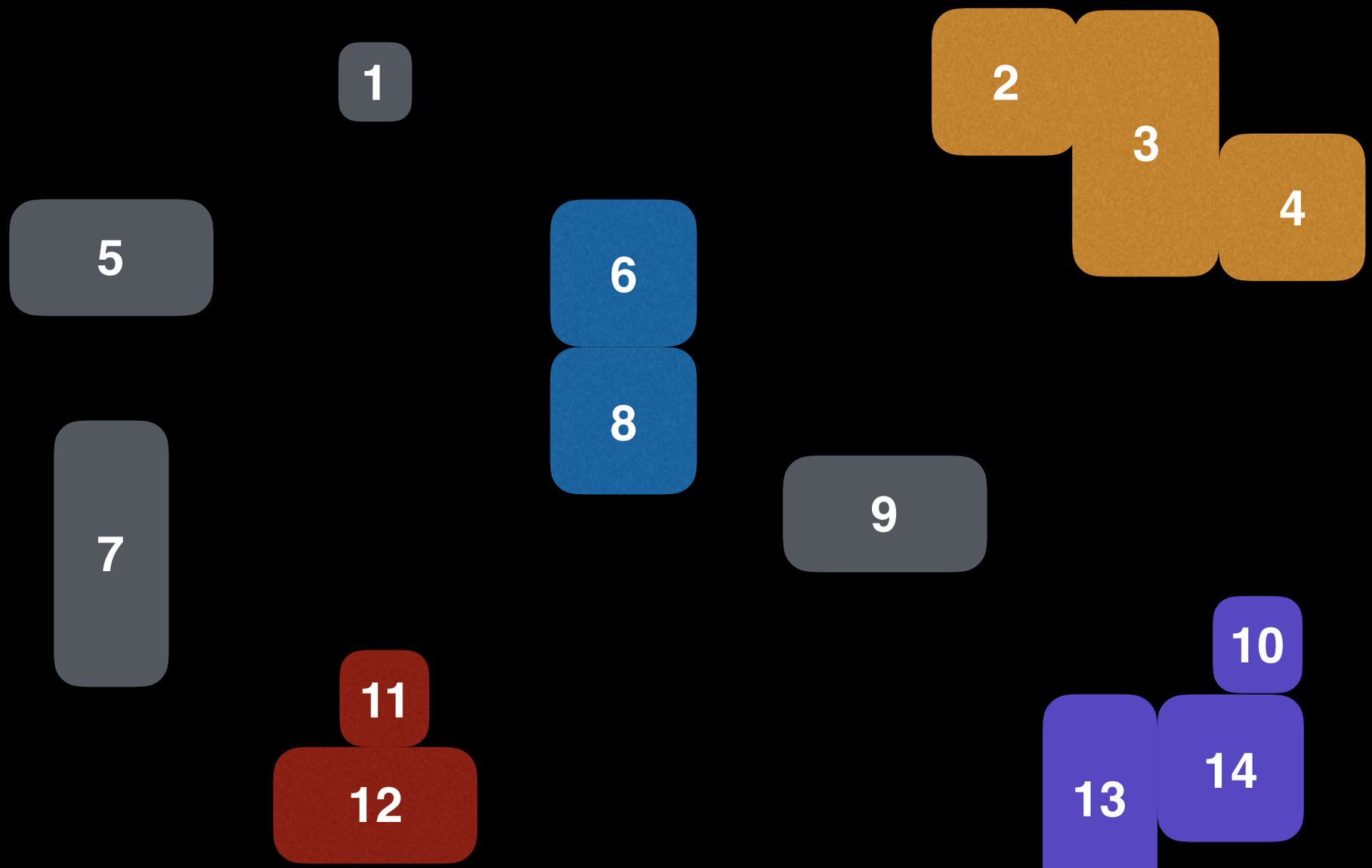
Union Find Magnets Example

Magnet 1
Magnet 2
Magnet 3
Magnet 4
Magnet 5
Magnet 6
Magnet 7
Magnet 8
Magnet 9
Magnet 10
Magnet 11
Magnet 12
Magnet 13
Magnet 14



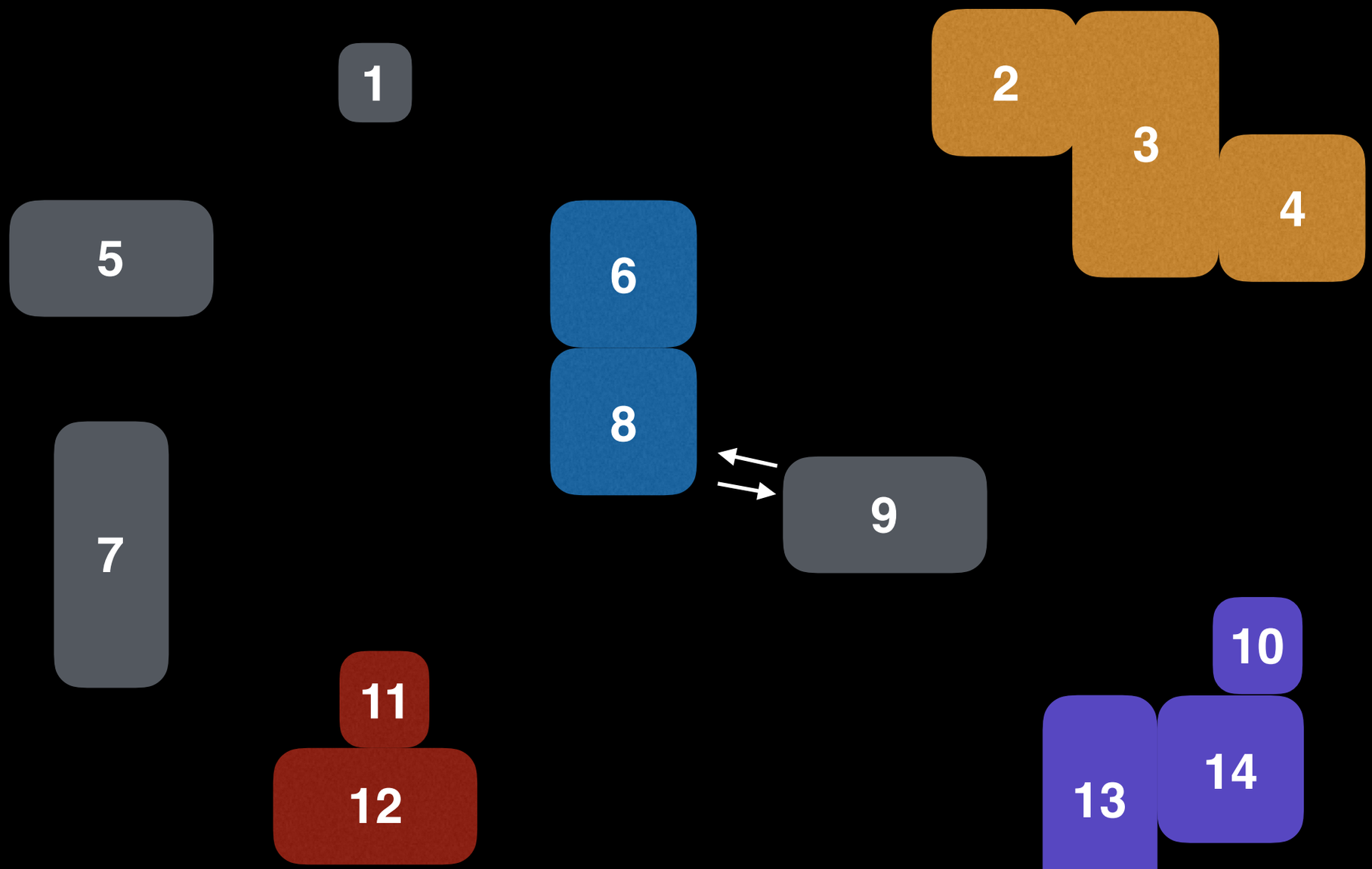
Union Find Magnets Example

Magnet 1
Magnet 2
Magnet 3
Magnet 4
Magnet 5
Magnet 6
Magnet 7
Magnet 8
Magnet 9
Magnet 10
Magnet 11
Magnet 12
Magnet 13
Magnet 14



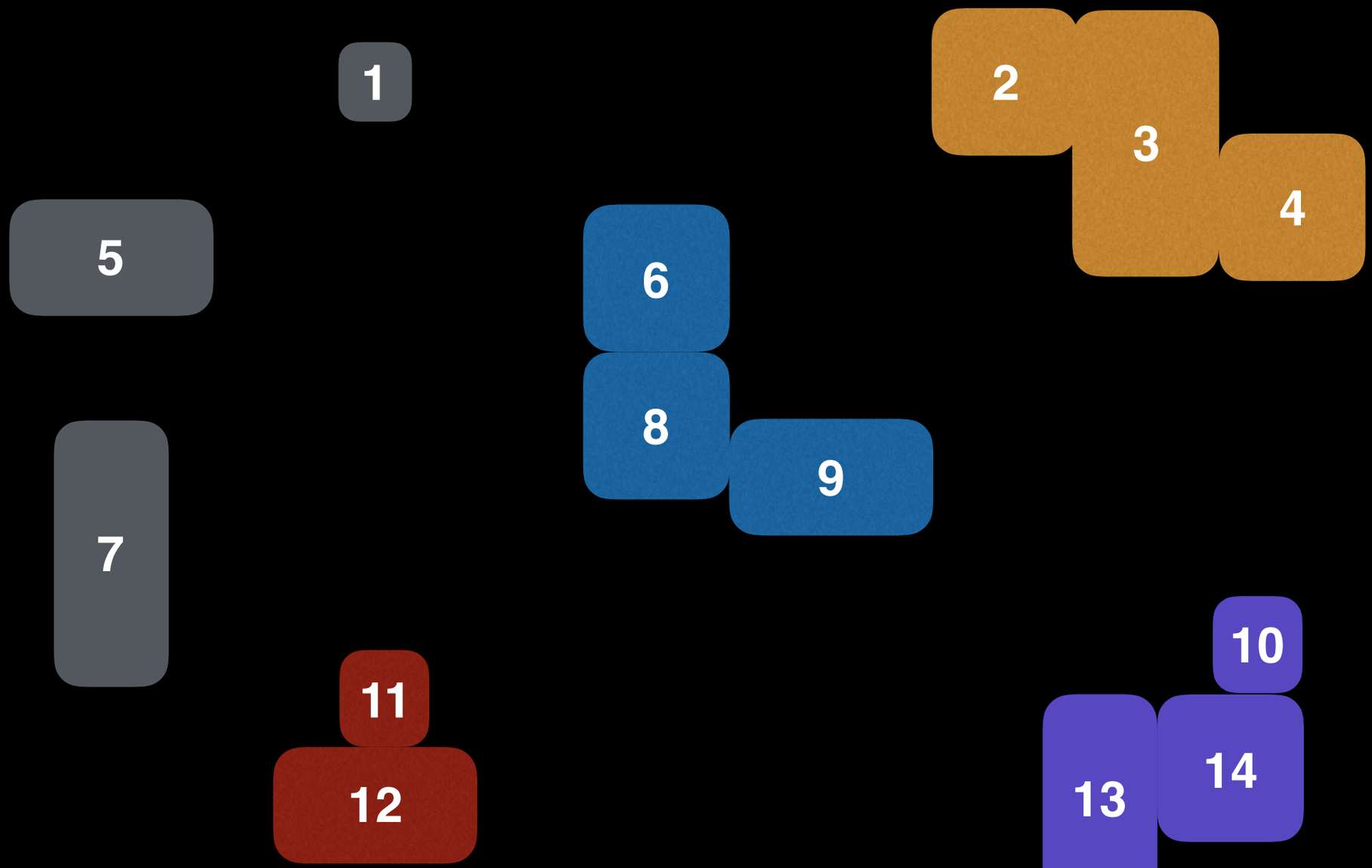
Union Find Magnets Example

Magnet 1
Magnet 2
Magnet 3
Magnet 4
Magnet 5
Magnet 6
Magnet 7
Magnet 8
Magnet 9
Magnet 10
Magnet 11
Magnet 12
Magnet 13
Magnet 14



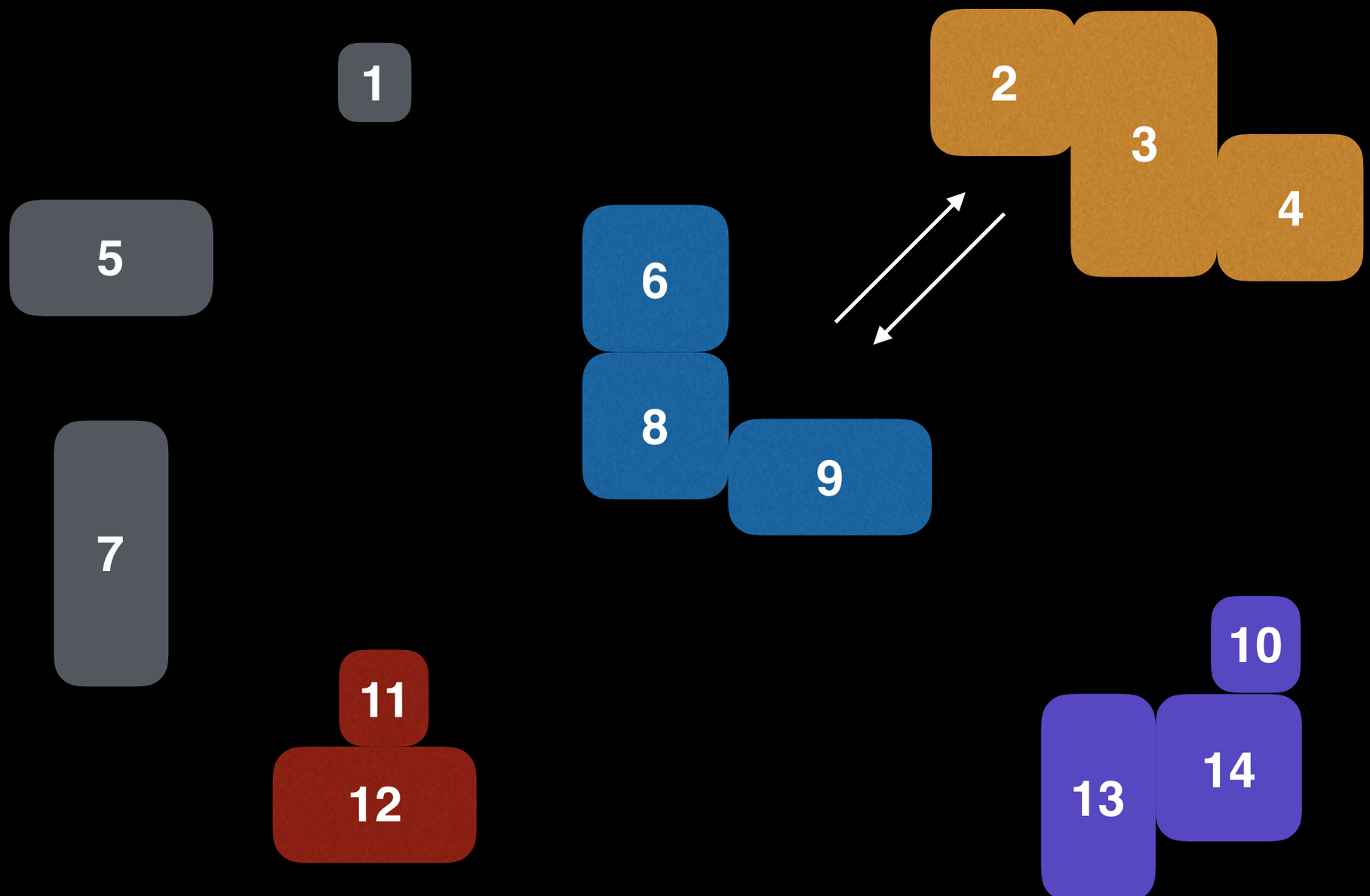
Union Find Magnets Example

Magnet 1
Magnet 2
Magnet 3
Magnet 4
Magnet 5
Magnet 6
Magnet 7
Magnet 8
Magnet 9
Magnet 10
Magnet 11
Magnet 12
Magnet 13
Magnet 14



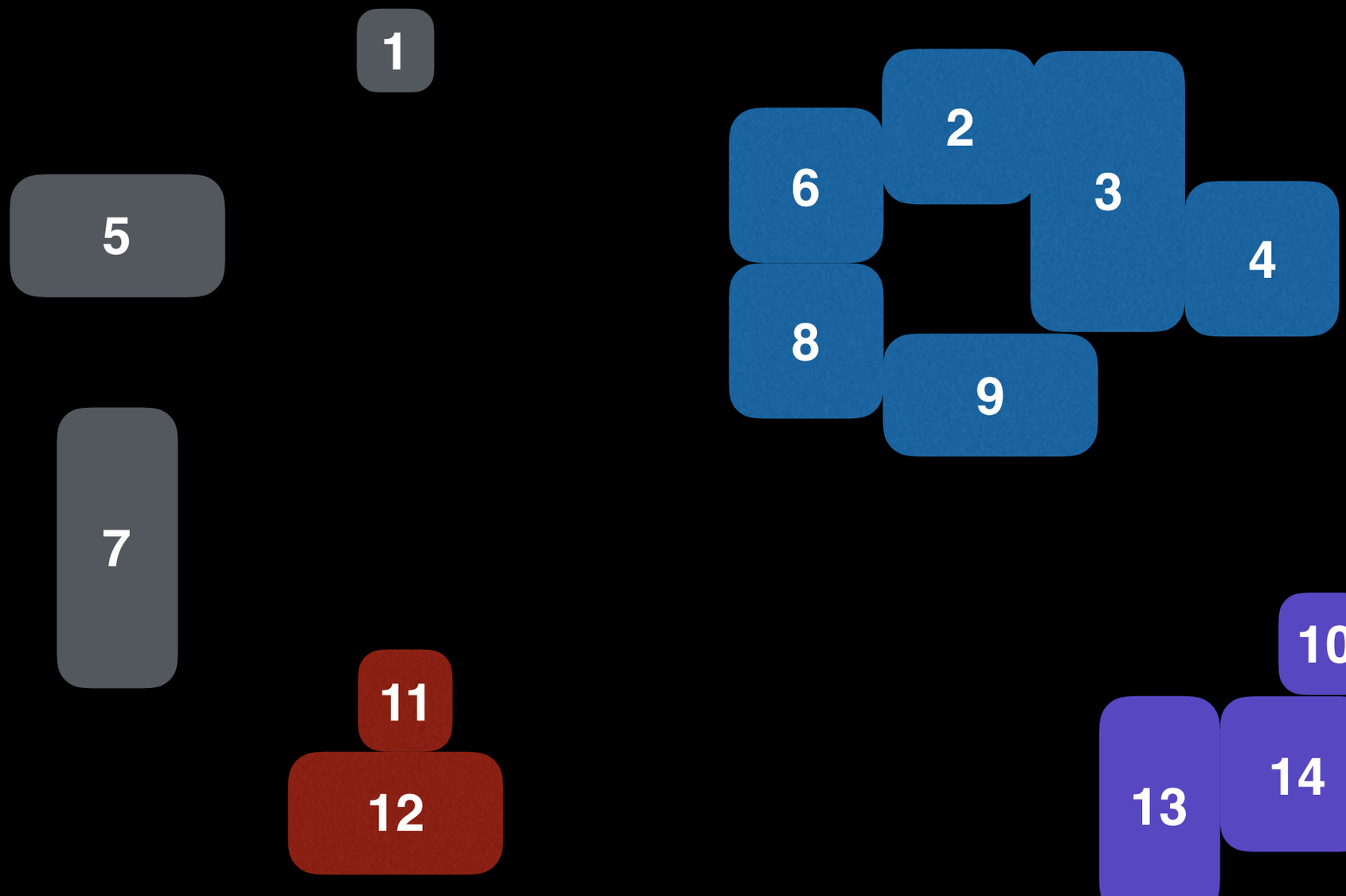
Union Find Magnets Example

Magnet 1
Magnet 2
Magnet 3
Magnet 4
Magnet 5
Magnet 6
Magnet 7
Magnet 8
Magnet 9
Magnet 10
Magnet 11
Magnet 12
Magnet 13
Magnet 14



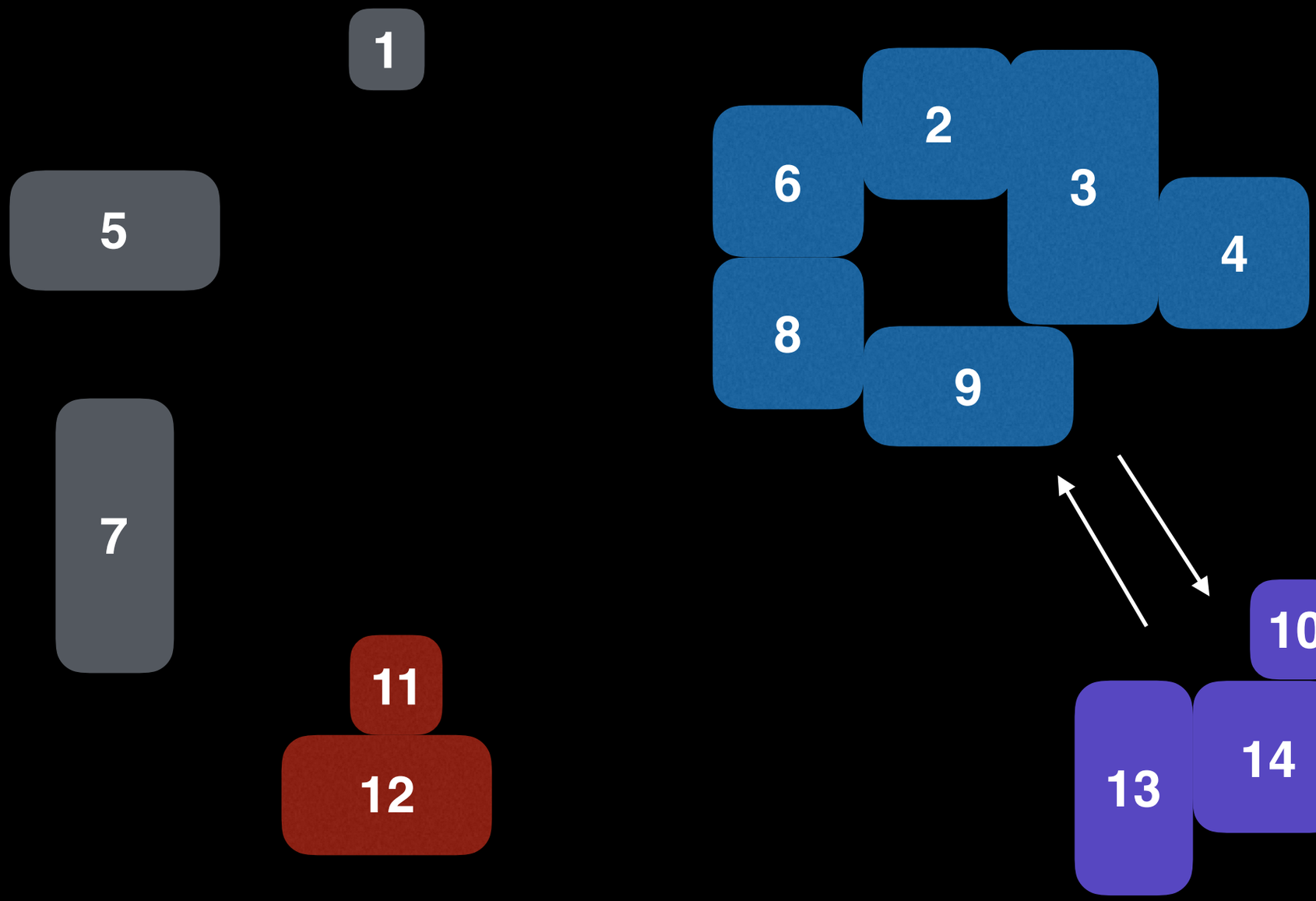
Union Find Magnets Example

Magnet 1
Magnet 2
Magnet 3
Magnet 4
Magnet 5
Magnet 6
Magnet 7
Magnet 8
Magnet 9
Magnet 10
Magnet 11
Magnet 12
Magnet 13
Magnet 14



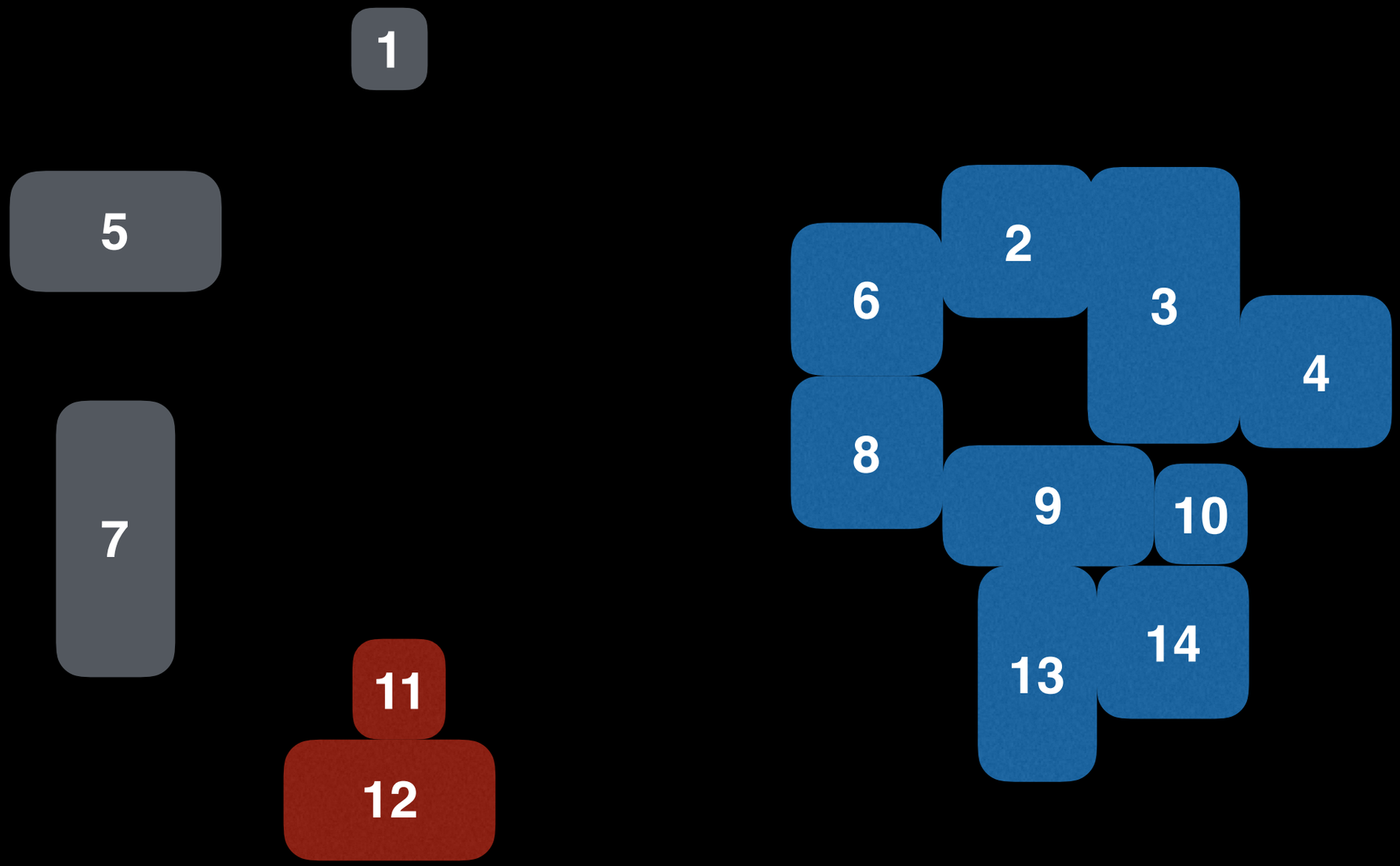
Union Find Magnets Example

Magnet 1
Magnet 2
Magnet 3
Magnet 4
Magnet 5
Magnet 6
Magnet 7
Magnet 8
Magnet 9
Magnet 10
Magnet 11
Magnet 12
Magnet 13
Magnet 14



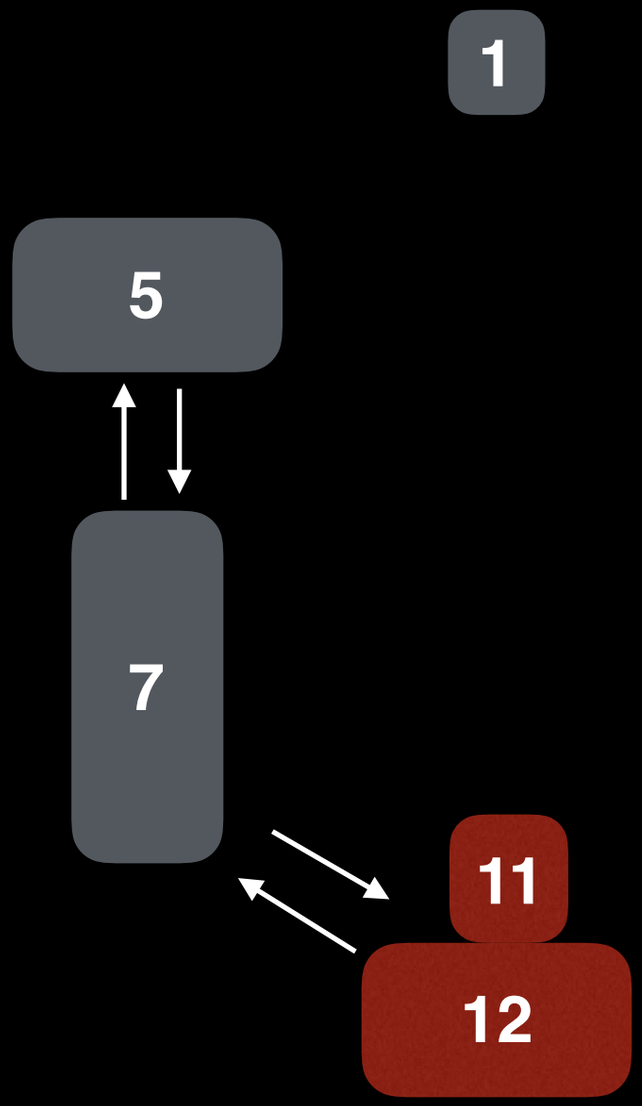
Union Find Magnets Example

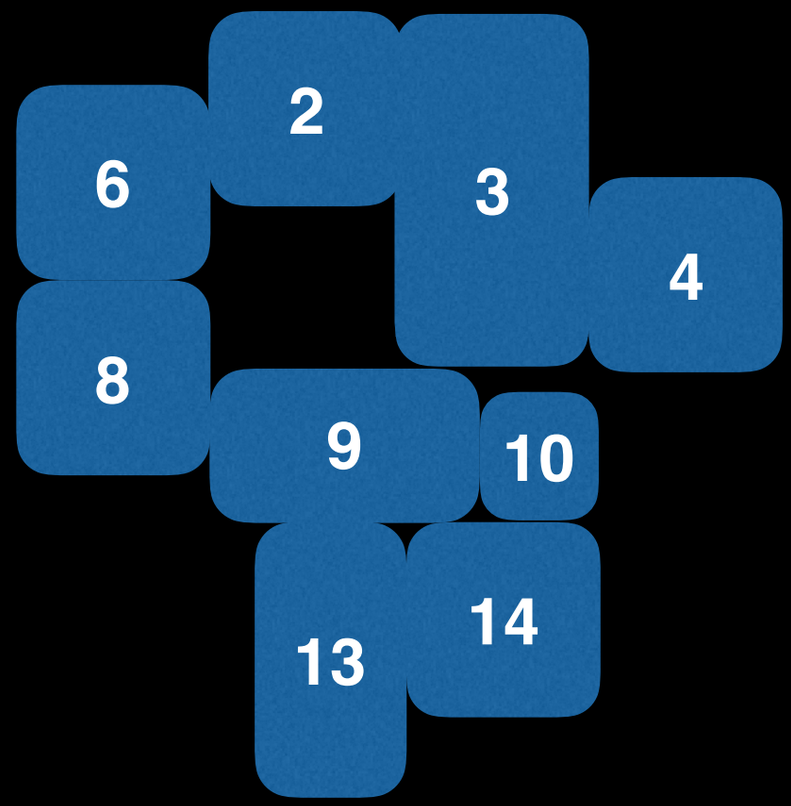
Magnet 1
Magnet 2
Magnet 3
Magnet 4
Magnet 5
Magnet 6
Magnet 7
Magnet 8
Magnet 9
Magnet 10
Magnet 11
Magnet 12
Magnet 13
Magnet 14



Union Find Magnets Example

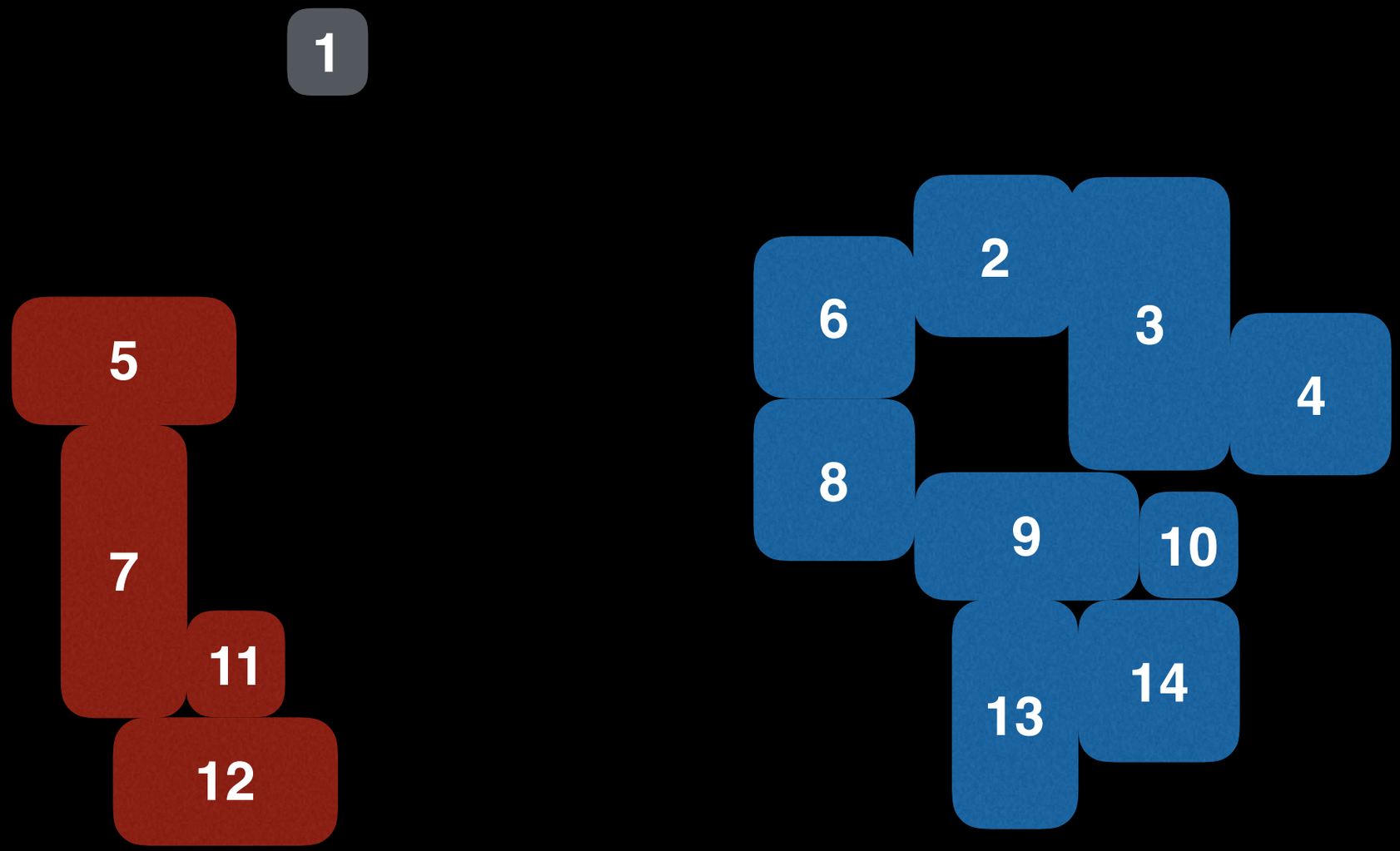
Magnet 1
Magnet 2
Magnet 3
Magnet 4
Magnet 5
Magnet 6
Magnet 7
Magnet 8
Magnet 9
Magnet 10
Magnet 11
Magnet 12
Magnet 13
Magnet 14





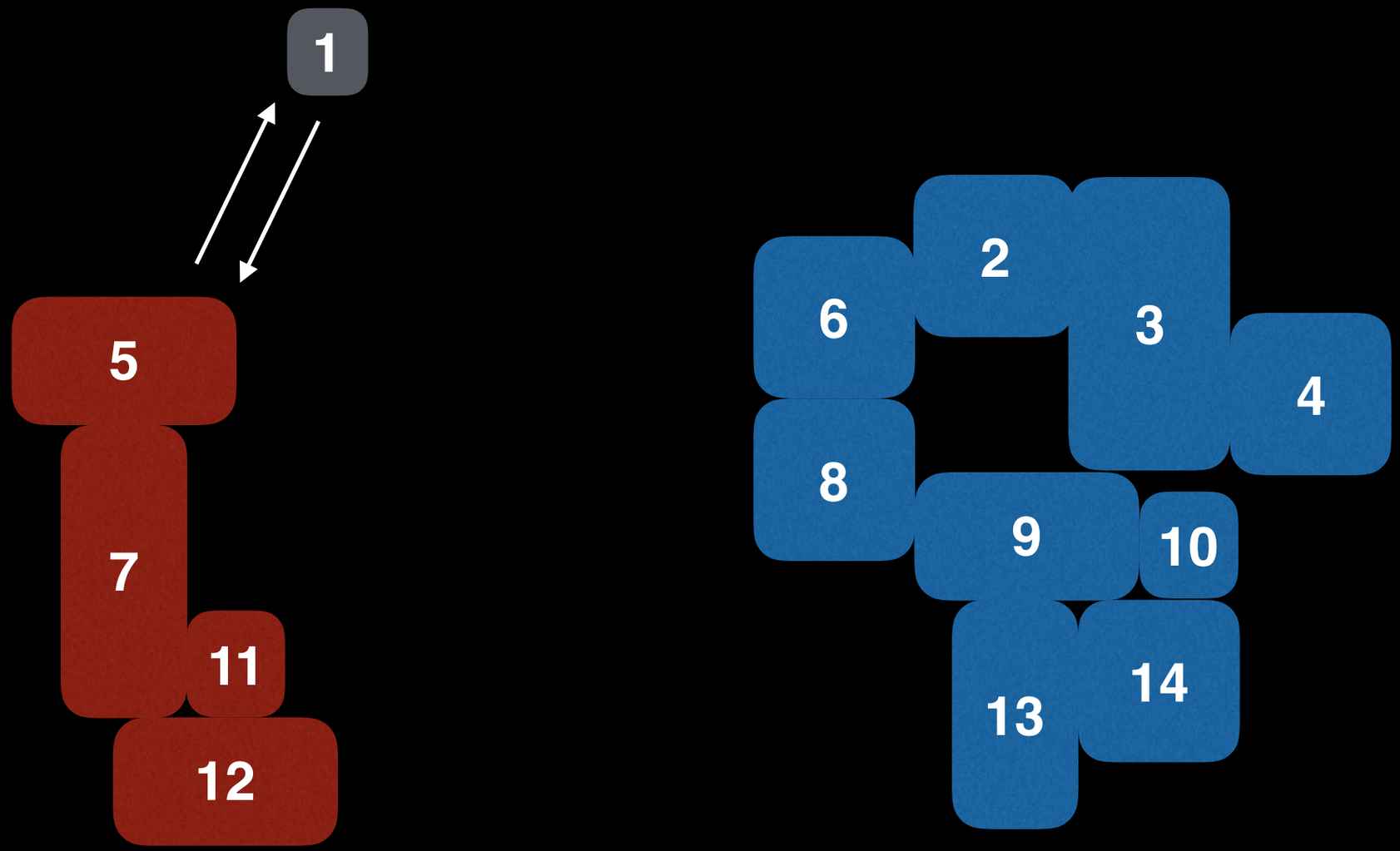
Union Find Magnets Example

Magnet 1
Magnet 2
Magnet 3
Magnet 4
Magnet 5
Magnet 6
Magnet 7
Magnet 8
Magnet 9
Magnet 10
Magnet 11
Magnet 12
Magnet 13
Magnet 14



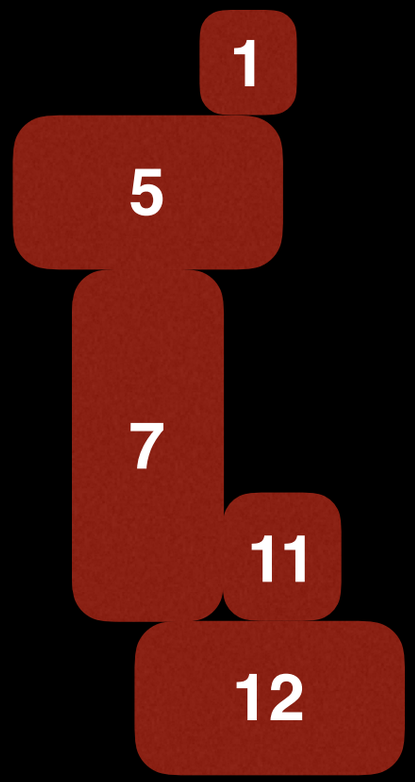
Union Find Magnets Example

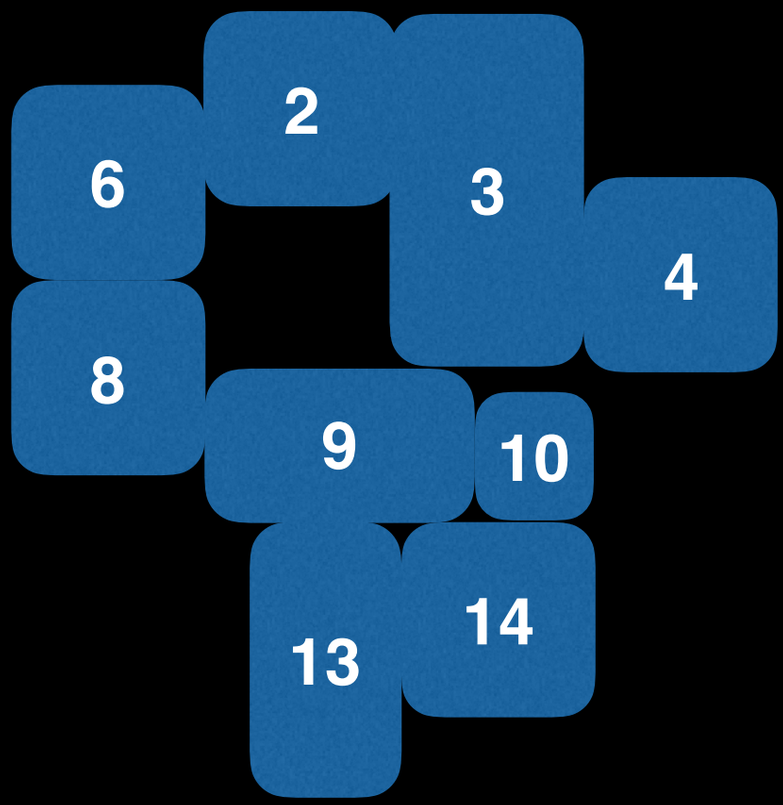
Magnet 1
Magnet 2
Magnet 3
Magnet 4
Magnet 5
Magnet 6
Magnet 7
Magnet 8
Magnet 9
Magnet 10
Magnet 11
Magnet 12
Magnet 13
Magnet 14



Union Find Magnets Example

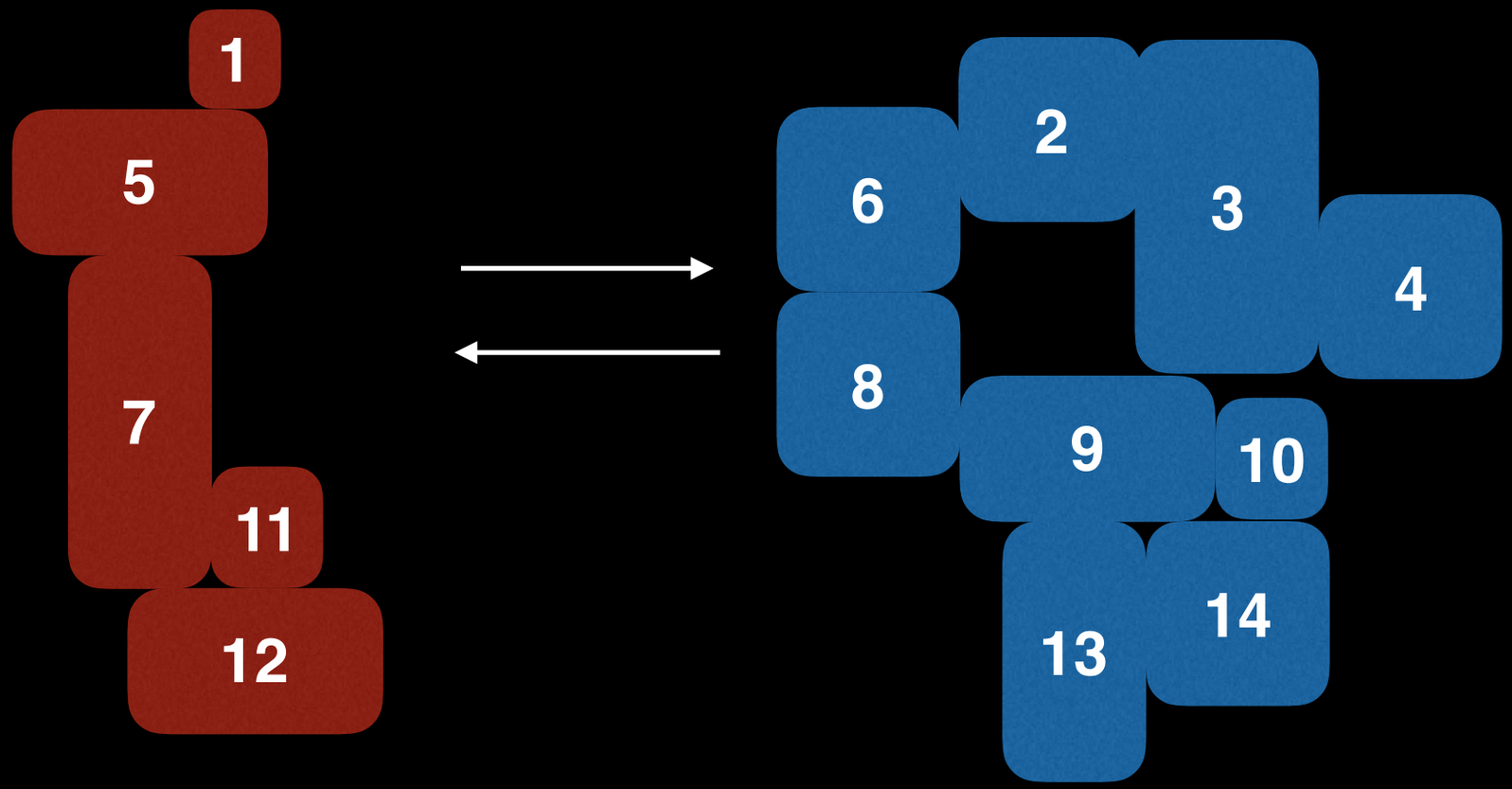
Magnet 1
Magnet 2
Magnet 3
Magnet 4
Magnet 5
Magnet 6
Magnet 7
Magnet 8
Magnet 9
Magnet 10
Magnet 11
Magnet 12
Magnet 13
Magnet 14





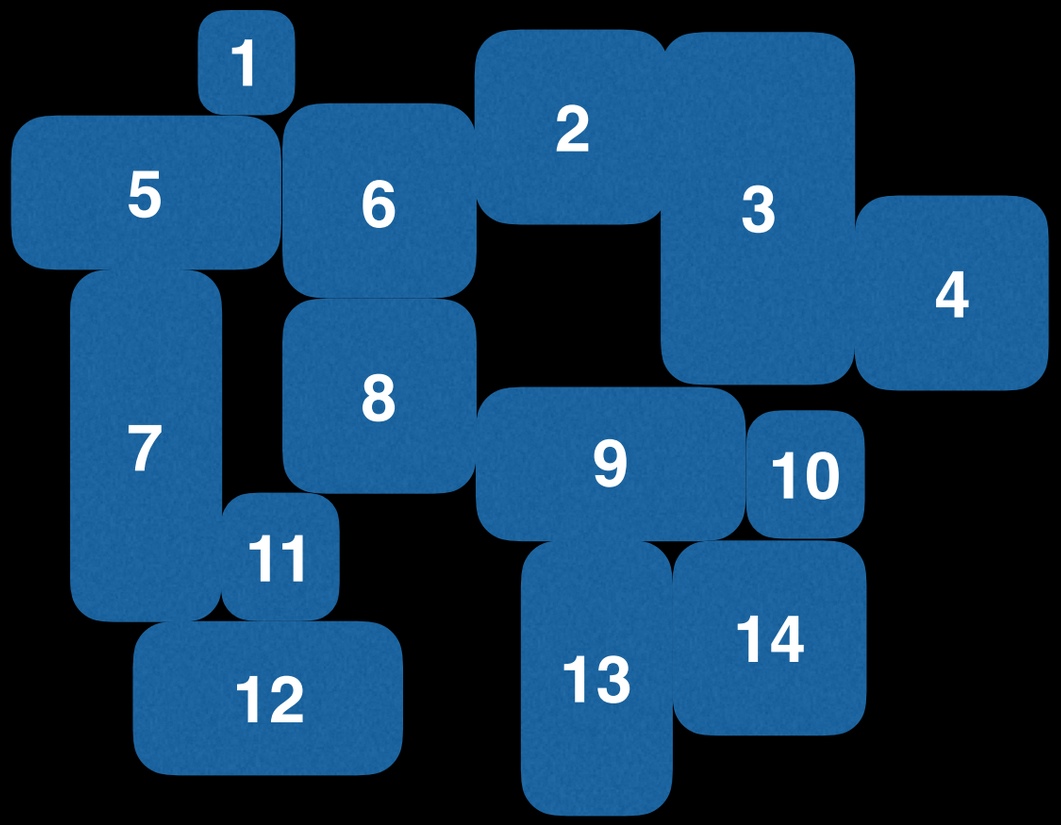
Union Find Magnets Example

Magnet 1
Magnet 2
Magnet 3
Magnet 4
Magnet 5
Magnet 6
Magnet 7
Magnet 8
Magnet 9
Magnet 10
Magnet 11
Magnet 12
Magnet 13
Magnet 14



Union Find Magnets Example

Magnet 1
Magnet 2
Magnet 3
Magnet 4
Magnet 5
Magnet 6
Magnet 7
Magnet 8
Magnet 9
Magnet 10
Magnet 11
Magnet 12
Magnet 13
Magnet 14



When and where is a Union Find used?

Kruskal's minimum spanning tree algorithm

Grid percolation

Network connectivity

Least common ancestor in trees

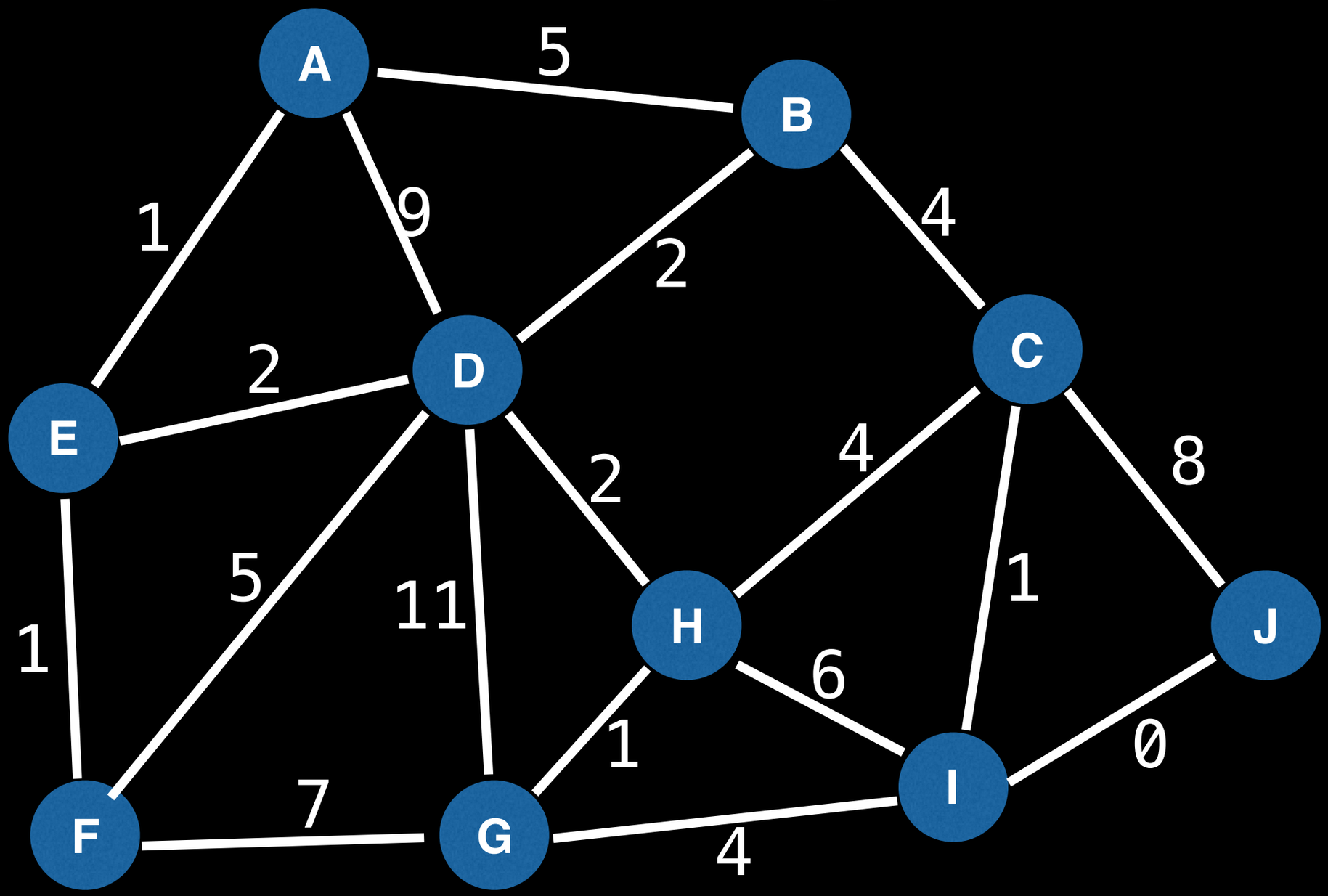
Image processing

Union Find application: Kruskal's Minimum Spanning Tree

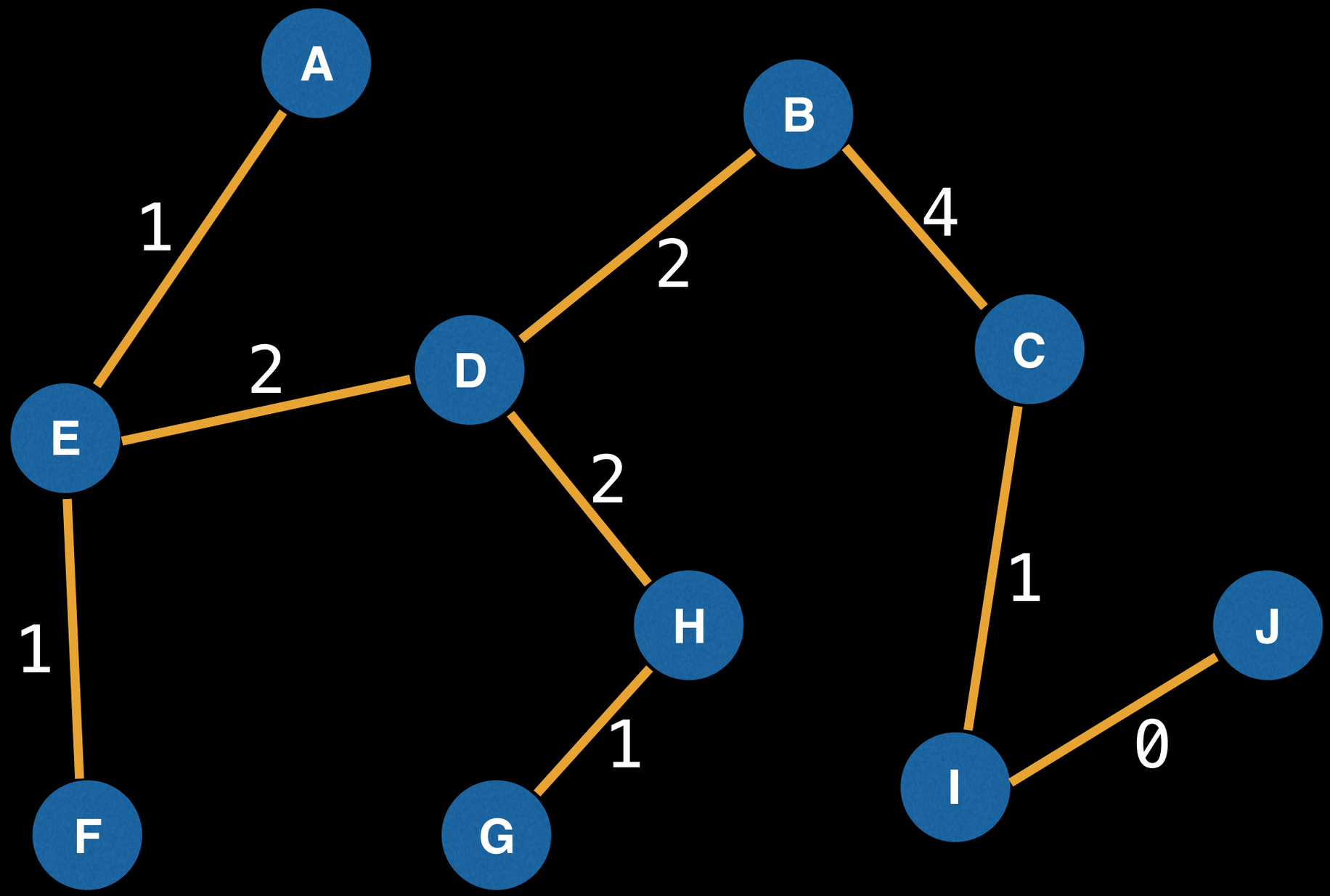
Given a graph G = (V,E) we want to find a **Minimum Spanning Tree** in the graph (it may not be unique).

A minimum spanning tree is a subset of the edges which connect all vertices in the graph with the minimal total edge cost.

Union Find application: Kruskal's Minimum Spanning Tree



Union Find application: Kruskal's Minimum Spanning Tree



Minimum spanning tree with weight 14

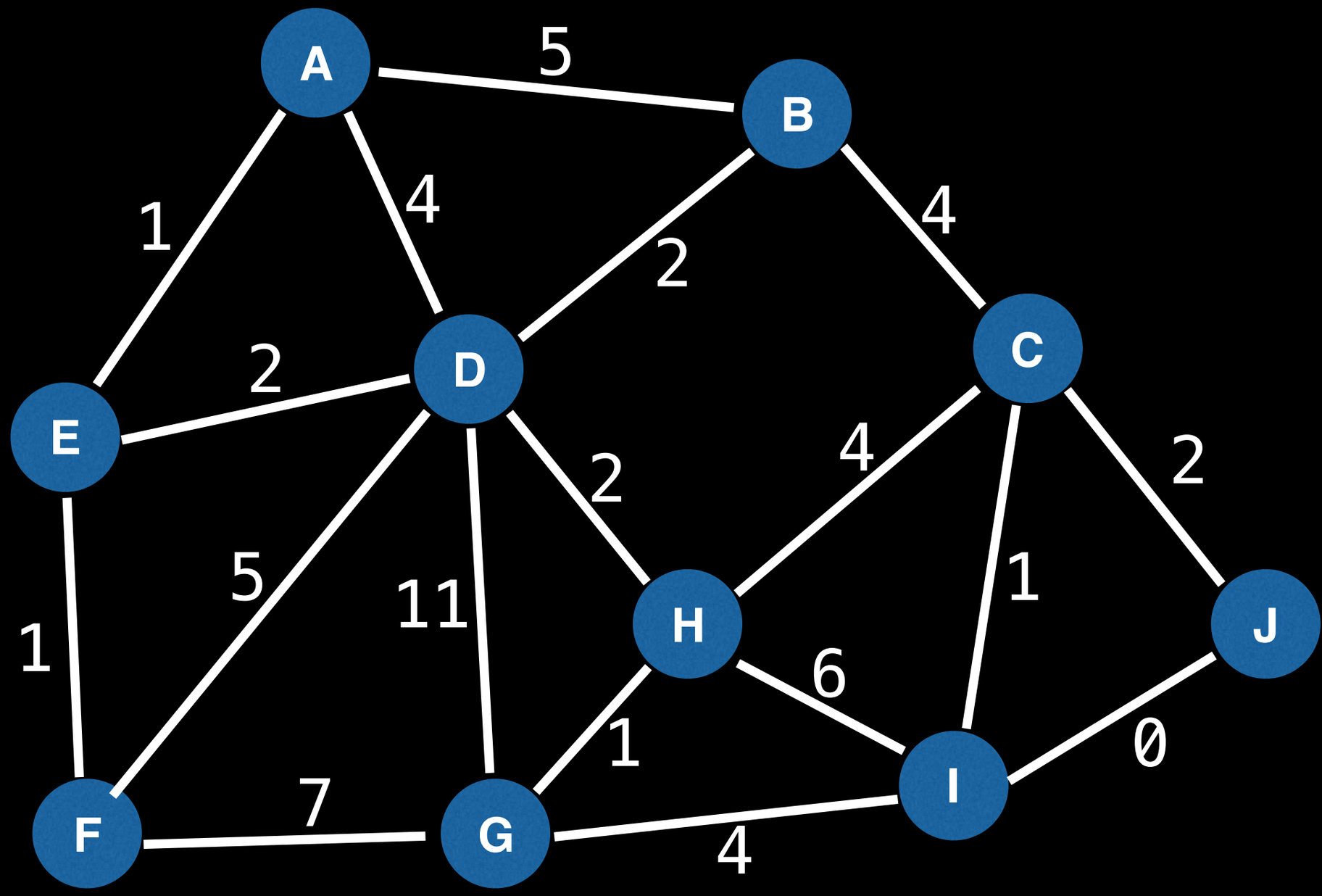
Union Find application: Kruskal's Minimum Spanning Tree

1) Sort edges by ascending edge weight.

2) Walk through the sorted edges, looking at the two nodes each edge belongs to. If they are already unified, we don't need include this edge. Otherwise we include it and unify the nodes.

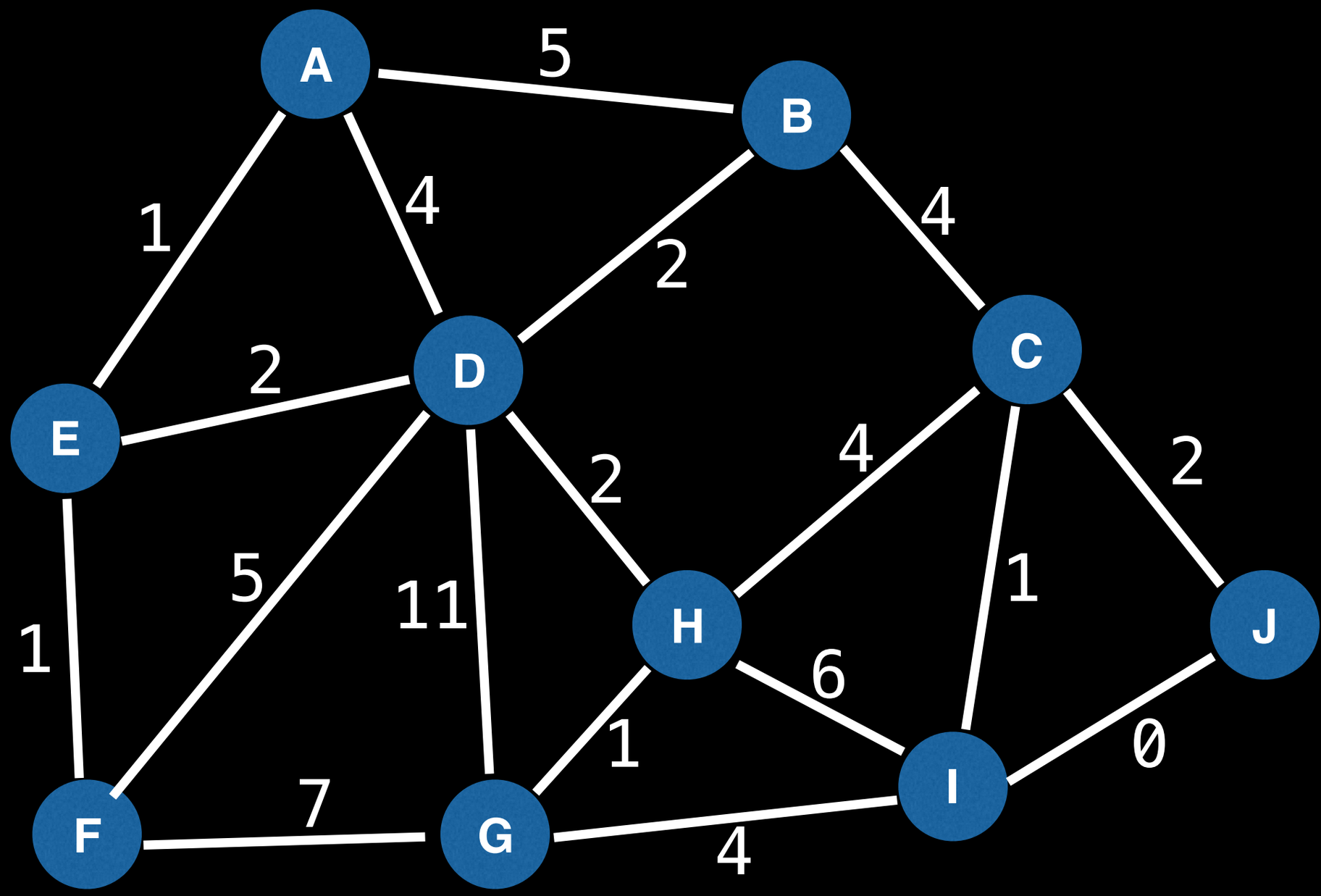
3) The algorithm terminates when every edge has been processed or all the vertices have been unified.

Union Find application: Kruskal's Minimum Spanning Tree



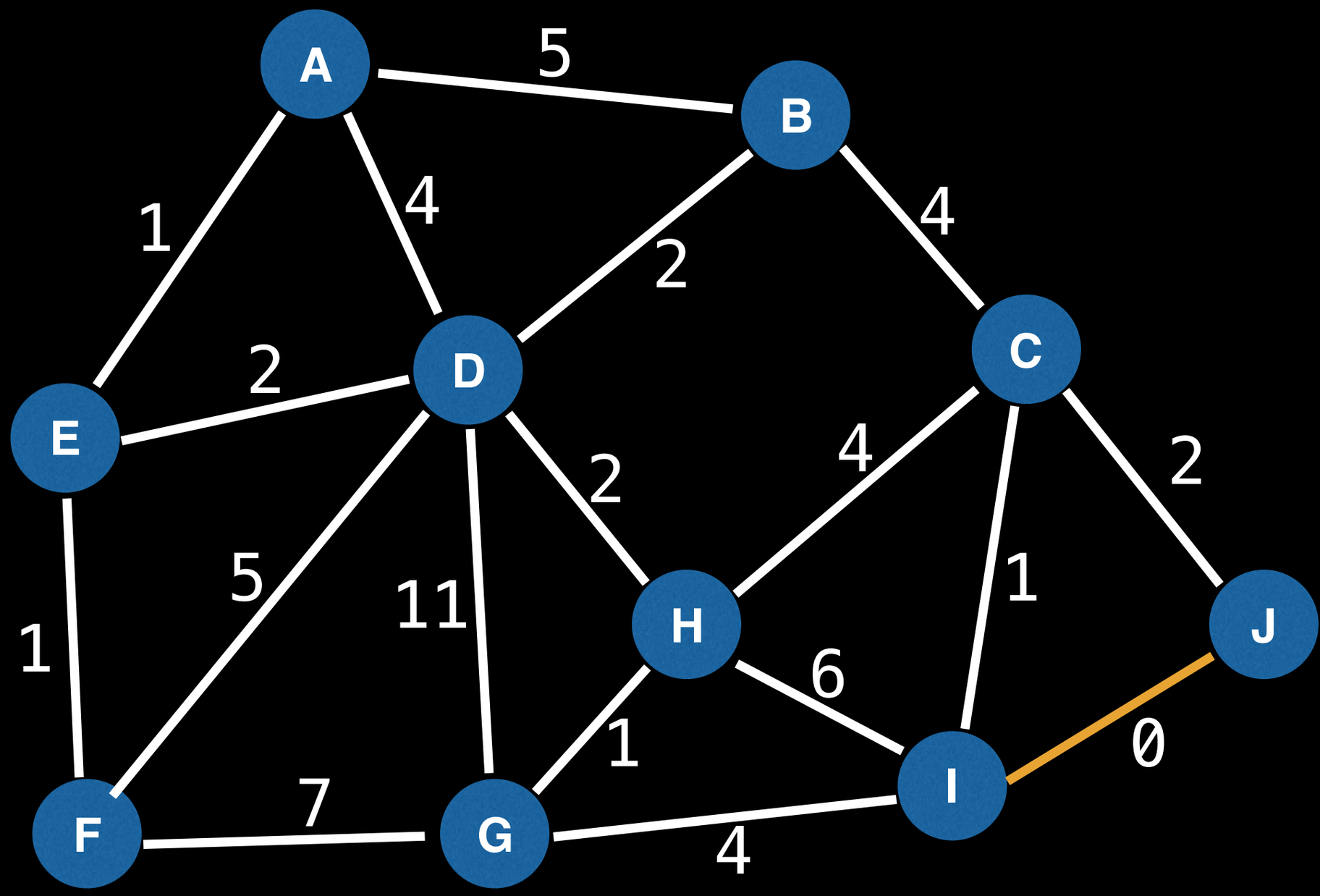
Union Find application: Kruskal's Minimum Spanning Tree

I to J = 0
A to E = 1
C to I = 1
E to F = 1
G to H = 1
B to D = 2
C to J = 2
D to E = 2
D to H = 2
A to D = 4
B to C = 4
C to H = 4
G to I = 4
A to B = 5
D to F = 5
H to I = 6
F to G = 7
D to G = 11



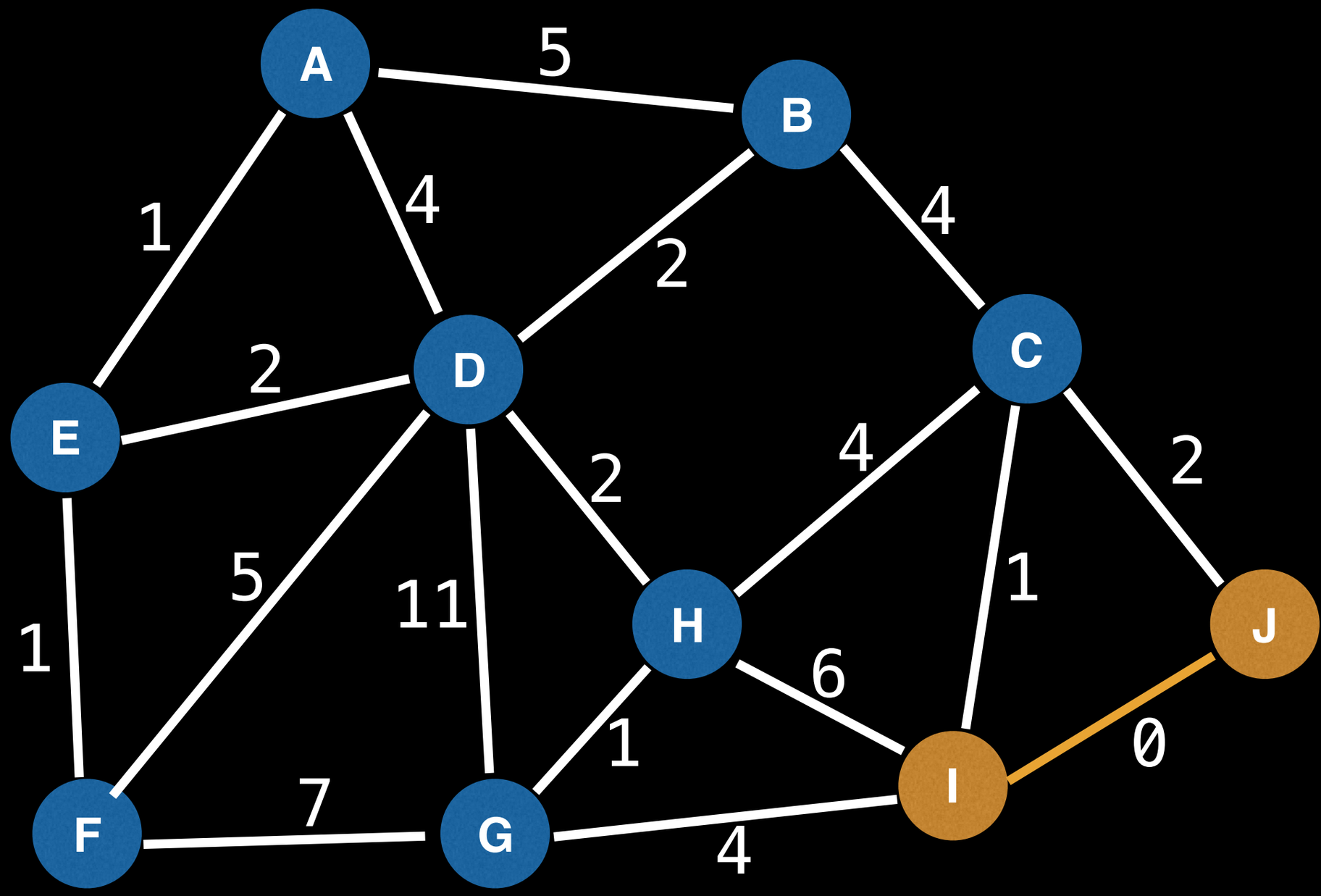
Union Find application: Kruskal's Minimum Spanning Tree

I to J = 0
A to E = 1
C to I = 1
E to F = 1
G to H = 1
B to D = 2
C to J = 2
D to E = 2
D to H = 2
A to D = 4
B to C = 4
C to H = 4
G to I = 4
A to B = 5
D to F = 5
H to I = 6
F to G = 7
D to G = 11



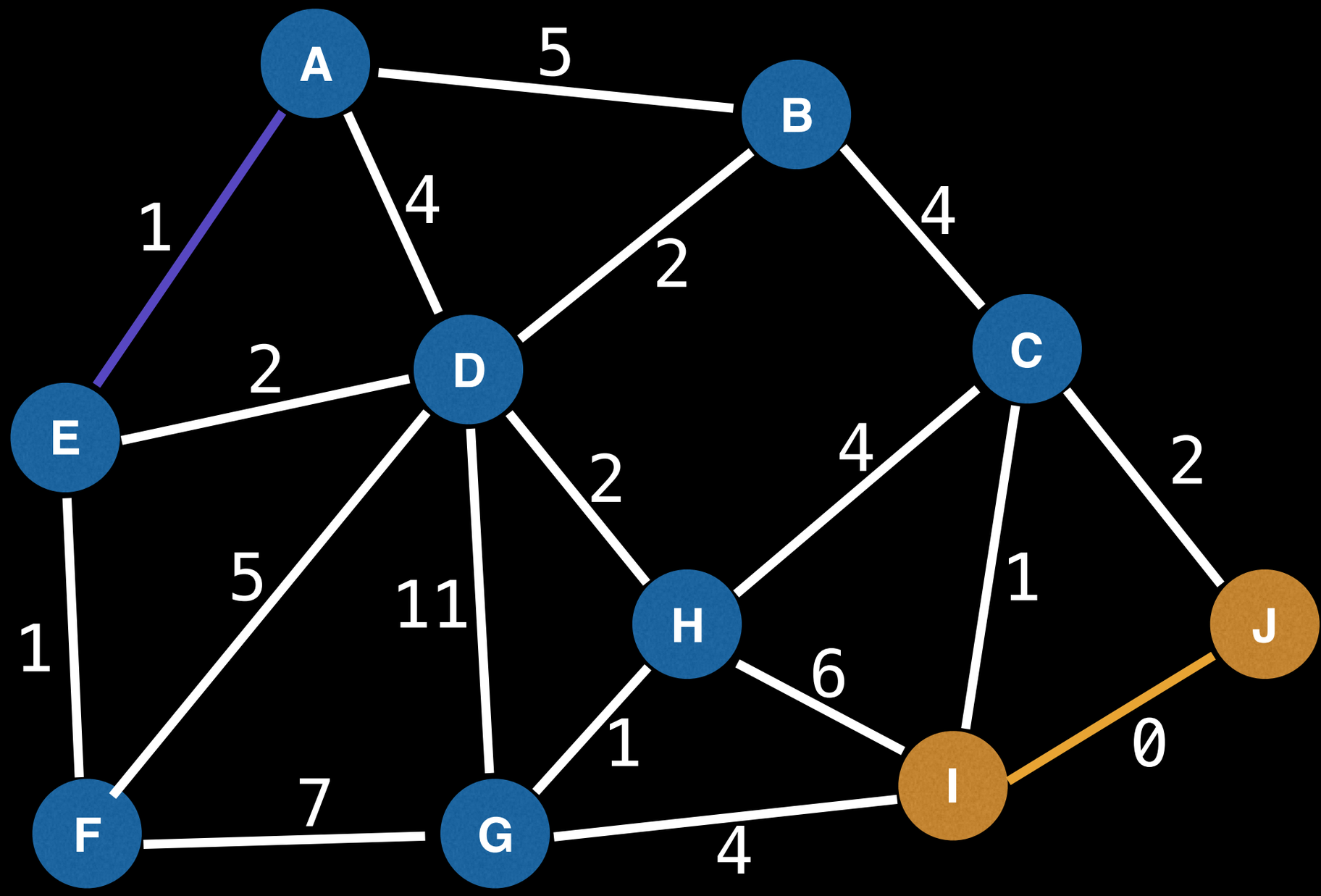
Union Find application: Kruskal's Minimum Spanning Tree

I to J = 0
A to E = 1
C to I = 1
E to F = 1
G to H = 1
B to D = 2
C to J = 2
D to E = 2
D to H = 2
A to D = 4
B to C = 4
C to H = 4
G to I = 4
A to B = 5
D to F = 5
H to I = 6
F to G = 7
D to G = 11



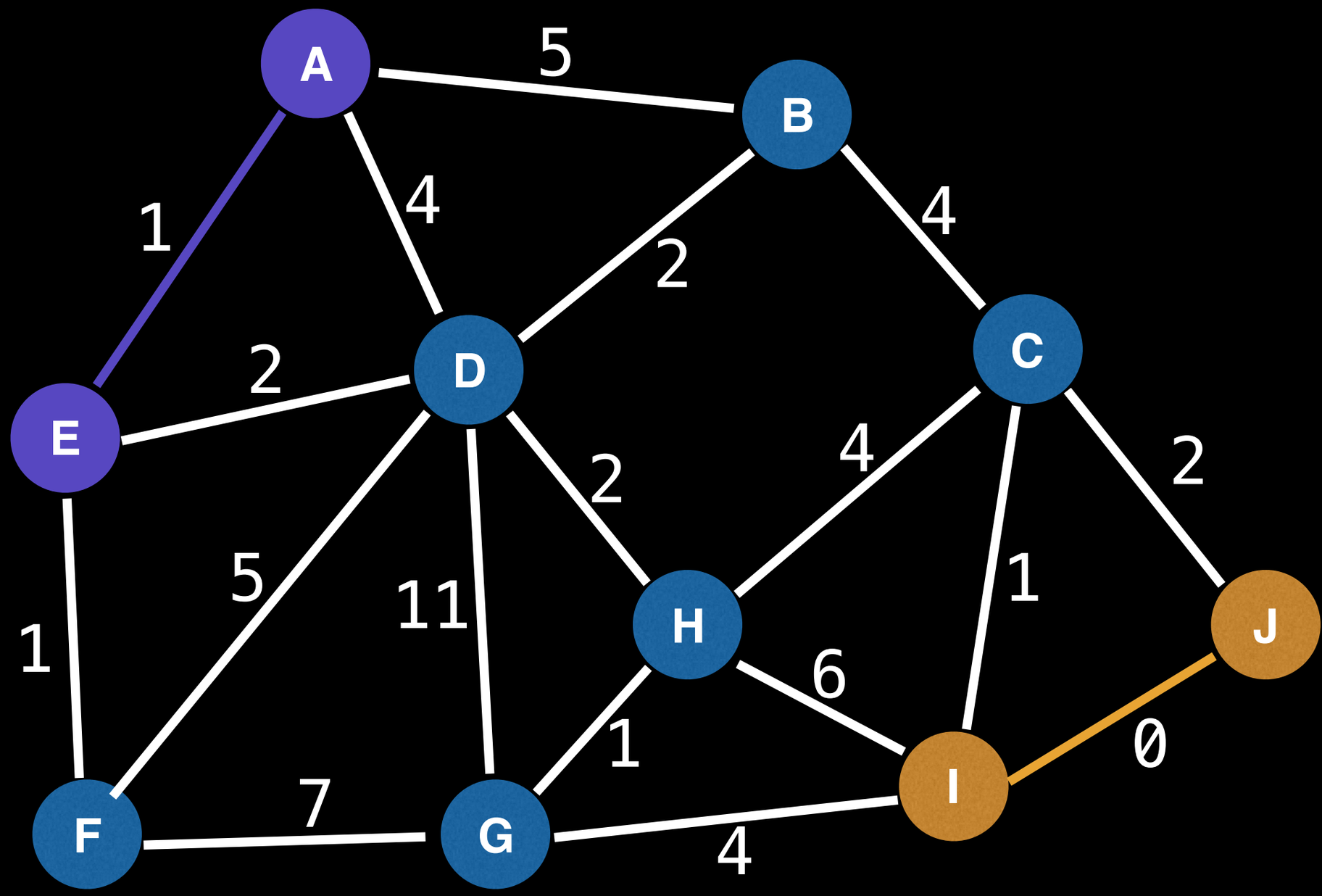
Union Find application: Kruskal's Minimum Spanning Tree

I to J = 0
A to E = 1
C to I = 1
E to F = 1
G to H = 1
B to D = 2
C to J = 2
D to E = 2
D to H = 2
A to D = 4
B to C = 4
C to H = 4
G to I = 4
A to B = 5
D to F = 5
H to I = 6
F to G = 7
D to G = 11



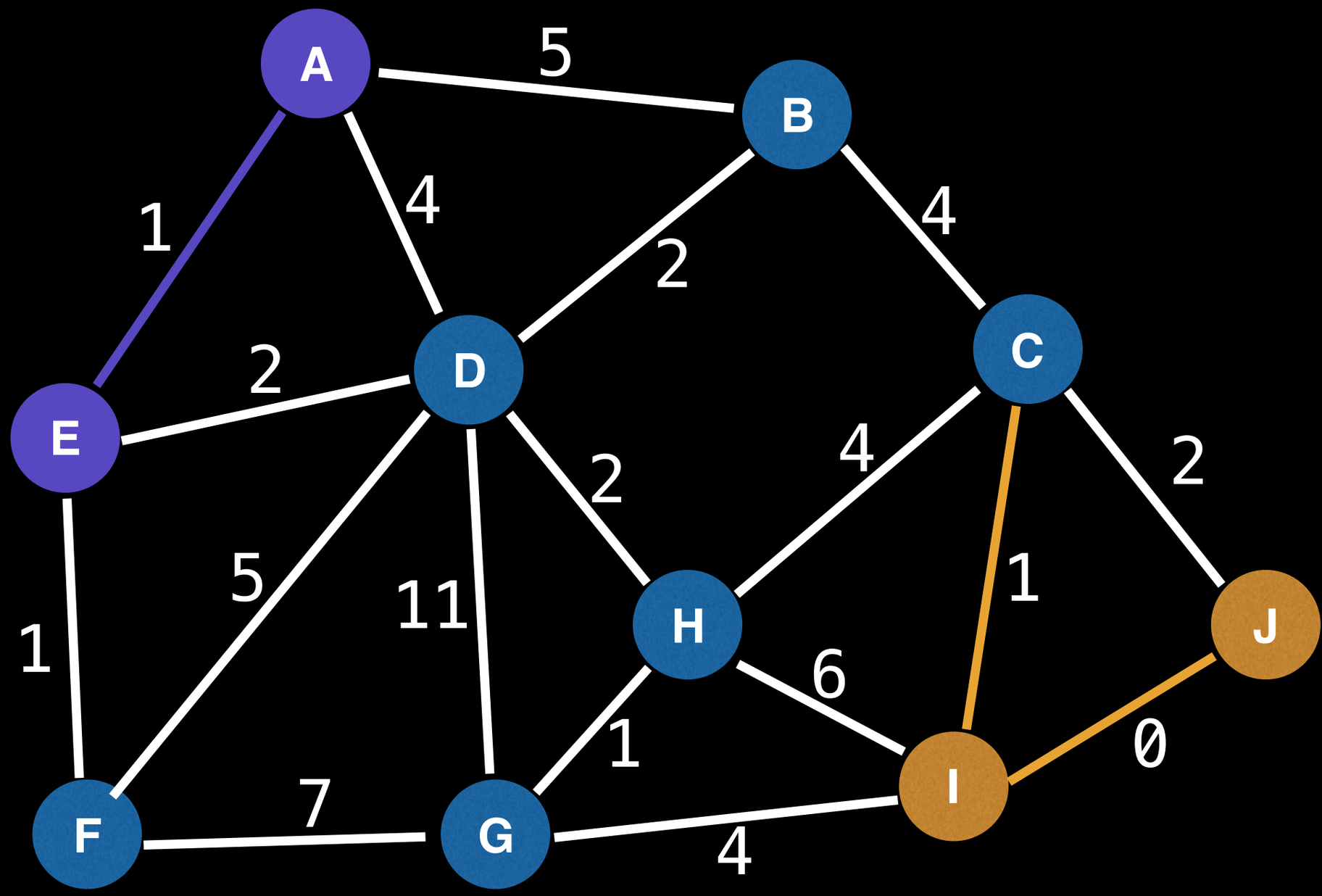
Union Find application: Kruskal's Minimum Spanning Tree

I to J = 0
A to E = 1
C to I = 1
E to F = 1
G to H = 1
B to D = 2
C to J = 2
D to E = 2
D to H = 2
A to D = 4
B to C = 4
C to H = 4
G to I = 4
A to B = 5
D to F = 5
H to I = 6
F to G = 7
D to G = 11



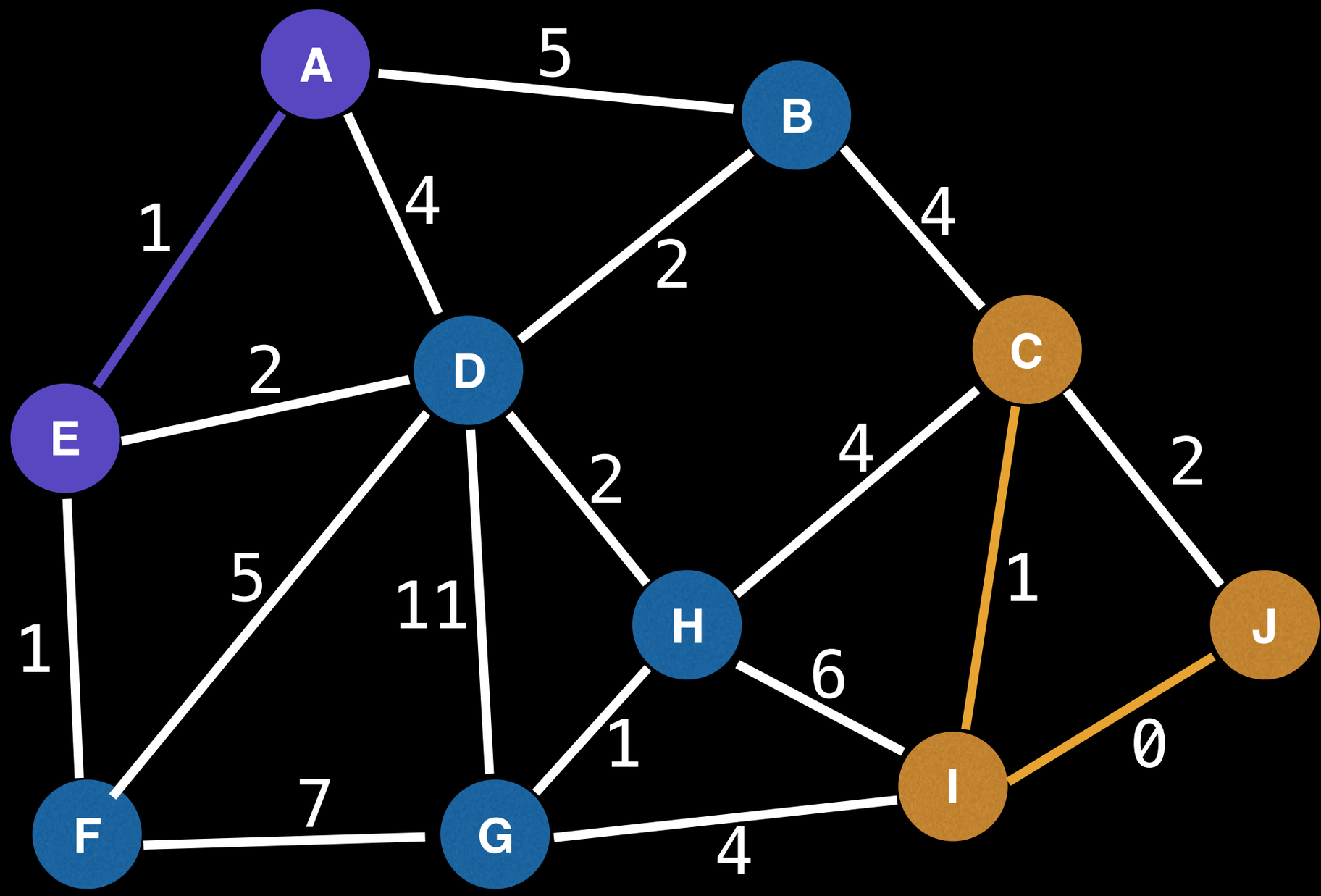
Union Find application: Kruskal's Minimum Spanning Tree

I to J = 0
A to E = 1
C to I = 1
E to F = 1
G to H = 1
B to D = 2
C to J = 2
D to E = 2
D to H = 2
A to D = 4
B to C = 4
C to H = 4
G to I = 4
A to B = 5
D to F = 5
H to I = 6
F to G = 7
D to G = 11



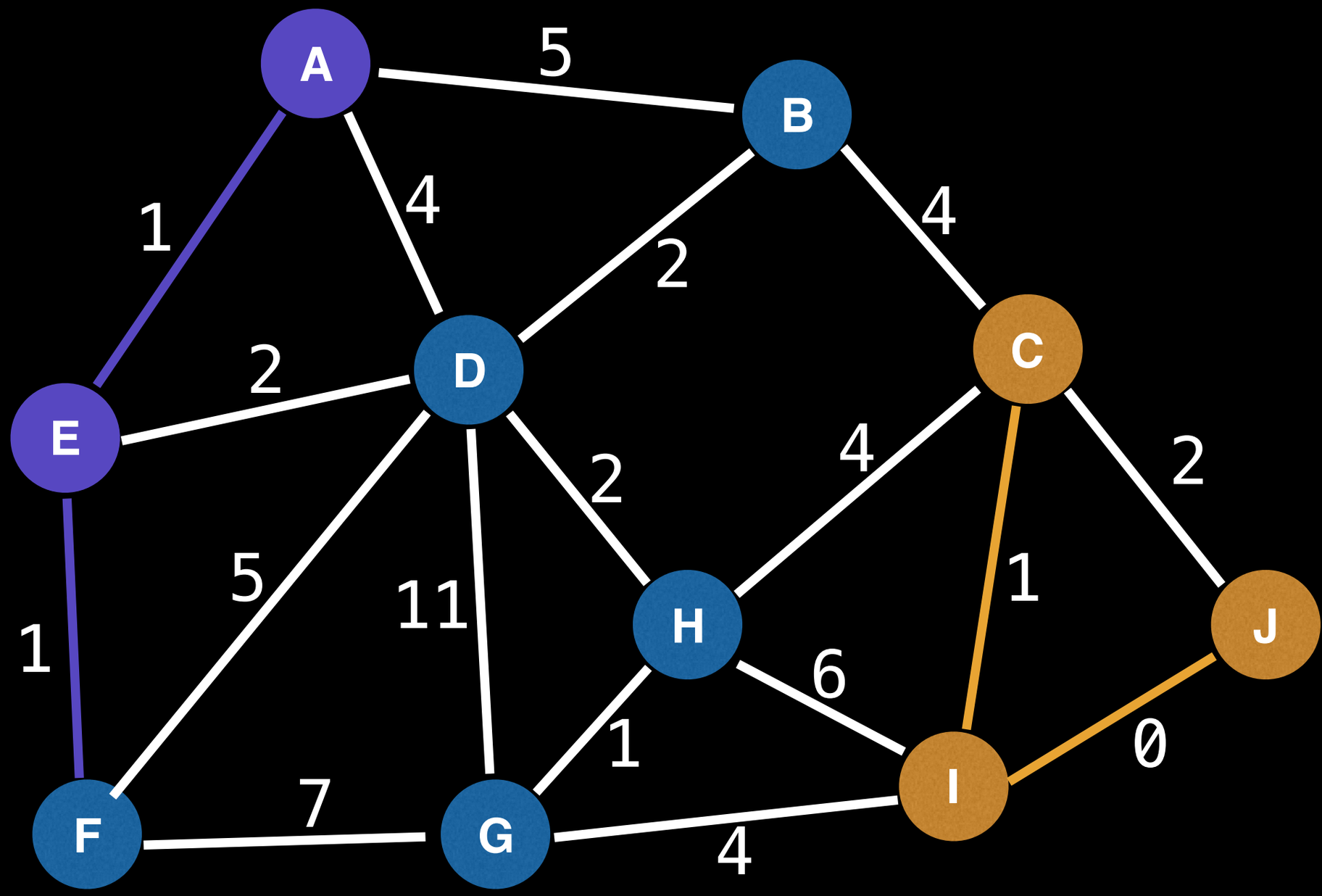
Union Find application: Kruskal's Minimum Spanning Tree

I to J = 0
A to E = 1
C to I = 1
E to F = 1
G to H = 1
B to D = 2
C to J = 2
D to E = 2
D to H = 2
A to D = 4
B to C = 4
C to H = 4
G to I = 4
A to B = 5
D to F = 5
H to I = 6
F to G = 7
D to G = 11



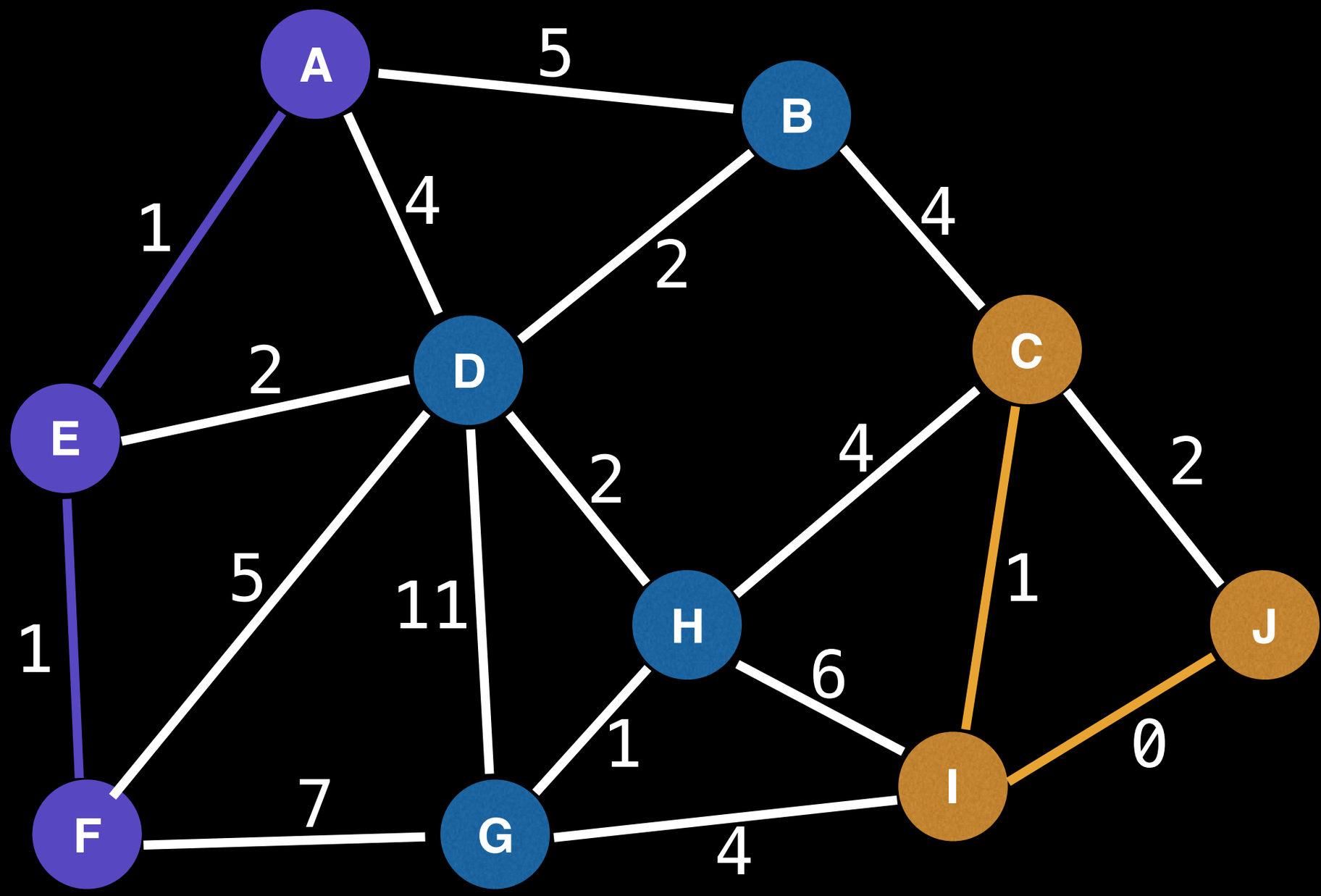
Union Find application: Kruskal's Minimum Spanning Tree

I to J = 0
A to E = 1
C to I = 1
E to F = 1
G to H = 1
B to D = 2
C to J = 2
D to E = 2
D to H = 2
A to D = 4
B to C = 4
C to H = 4
G to I = 4
A to B = 5
D to F = 5
H to I = 6
F to G = 7
D to G = 11



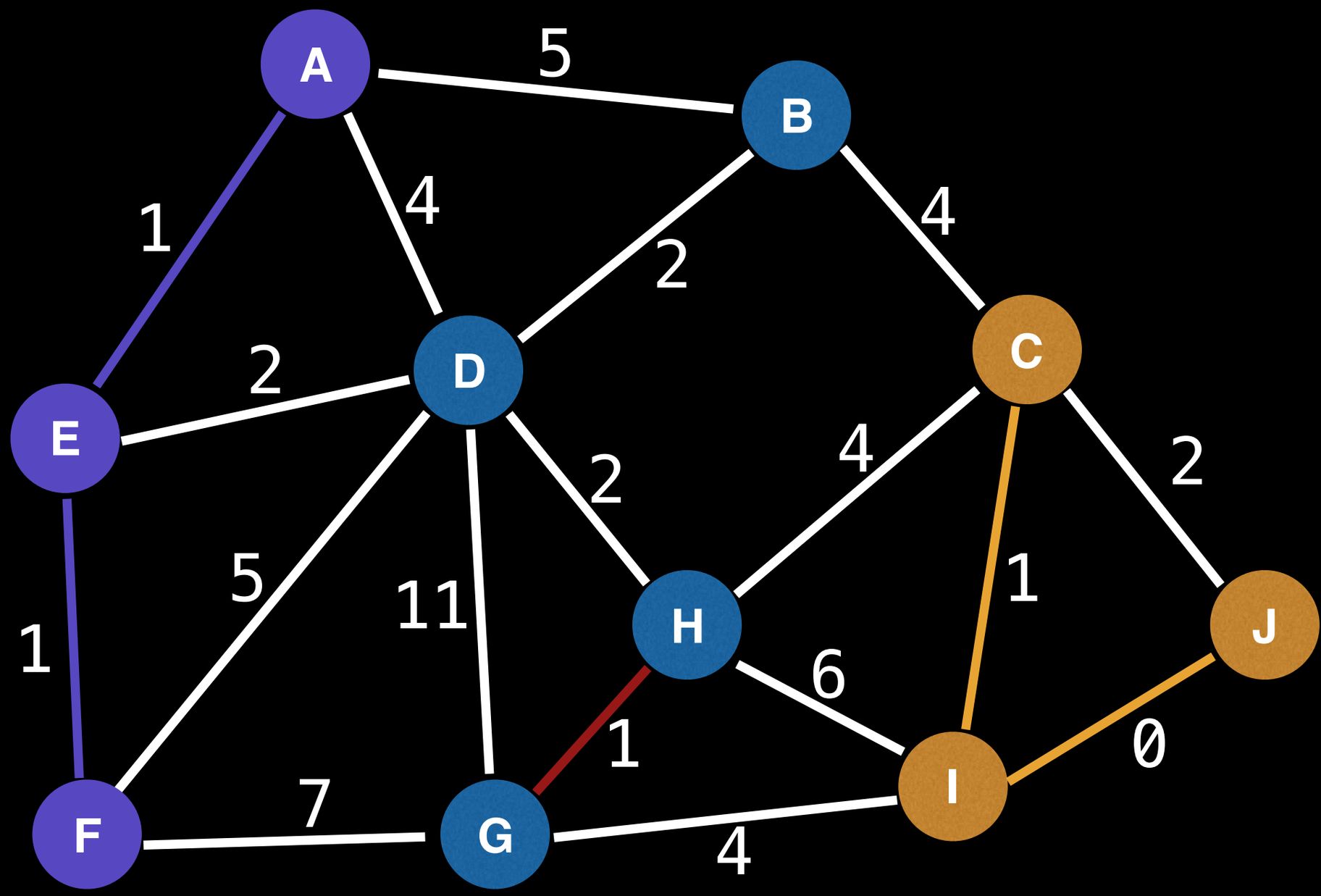
Union Find application: Kruskal's Minimum Spanning Tree

I to J = 0
A to E = 1
C to I = 1
E to F = 1
G to H = 1
B to D = 2
C to J = 2
D to E = 2
D to H = 2
A to D = 4
B to C = 4
C to H = 4
G to I = 4
A to B = 5
D to F = 5
H to I = 6
F to G = 7
D to G = 11



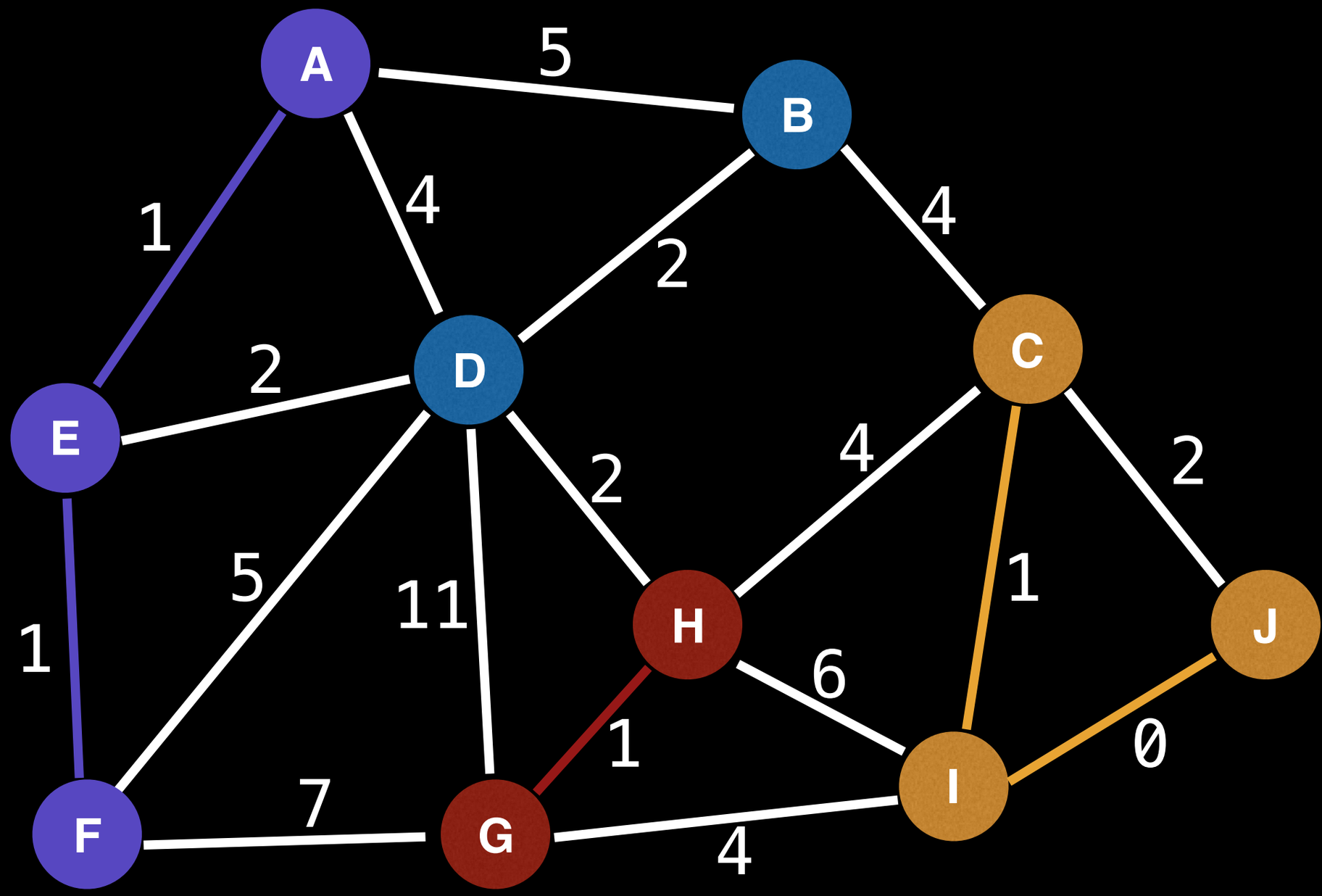
Union Find application: Kruskal's Minimum Spanning Tree

I to J = 0
A to E = 1
C to I = 1
E to F = 1
G to H = 1
B to D = 2
C to J = 2
D to E = 2
D to H = 2
A to D = 4
B to C = 4
C to H = 4
G to I = 4
A to B = 5
D to F = 5
H to I = 6
F to G = 7
D to G = 11



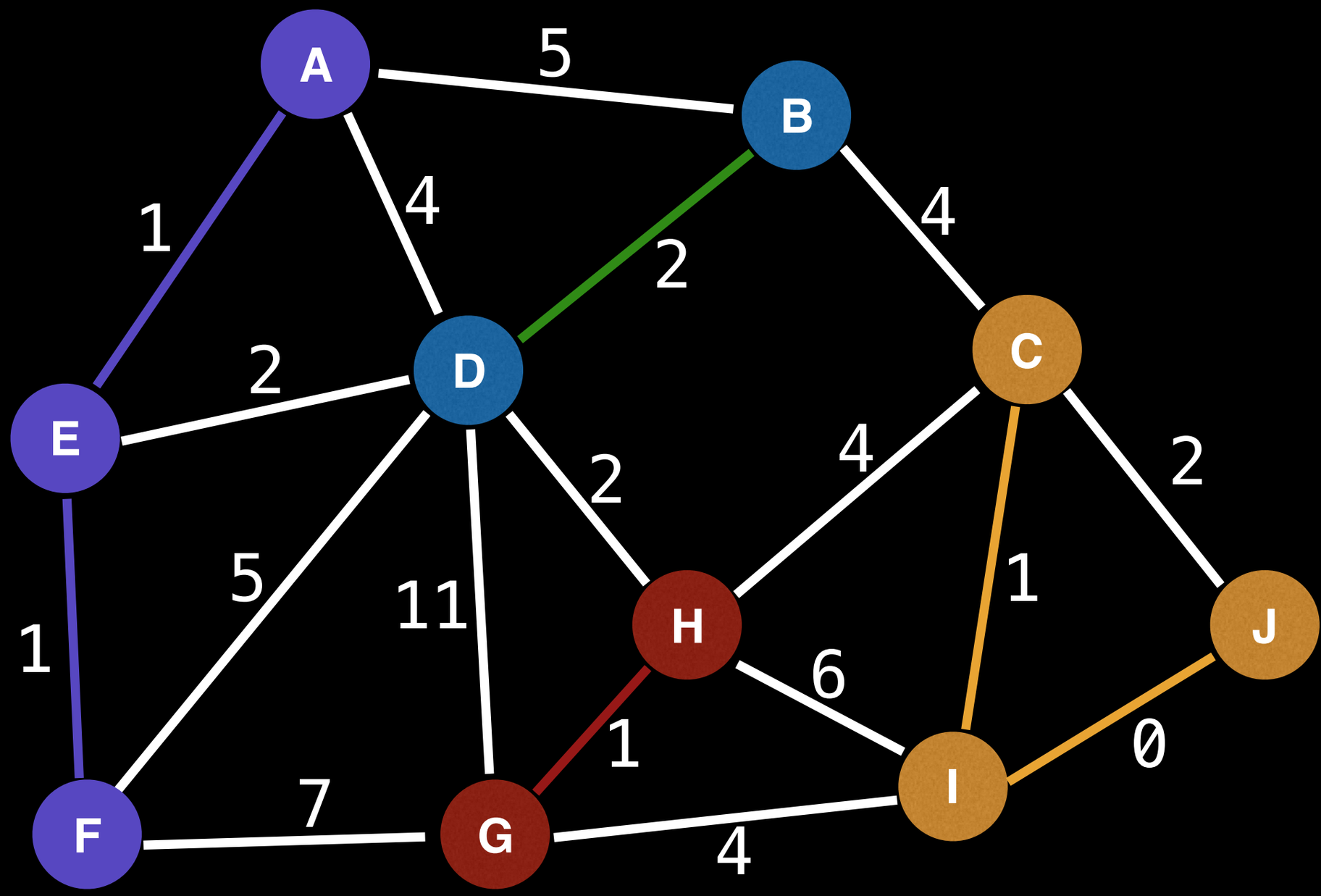
Union Find application: Kruskal's Minimum Spanning Tree

I to J = 0
A to E = 1
C to I = 1
E to F = 1
G to H = 1
B to D = 2
C to J = 2
D to E = 2
D to H = 2
A to D = 4
B to C = 4
C to H = 4
G to I = 4
A to B = 5
D to F = 5
H to I = 6
F to G = 7
D to G = 11



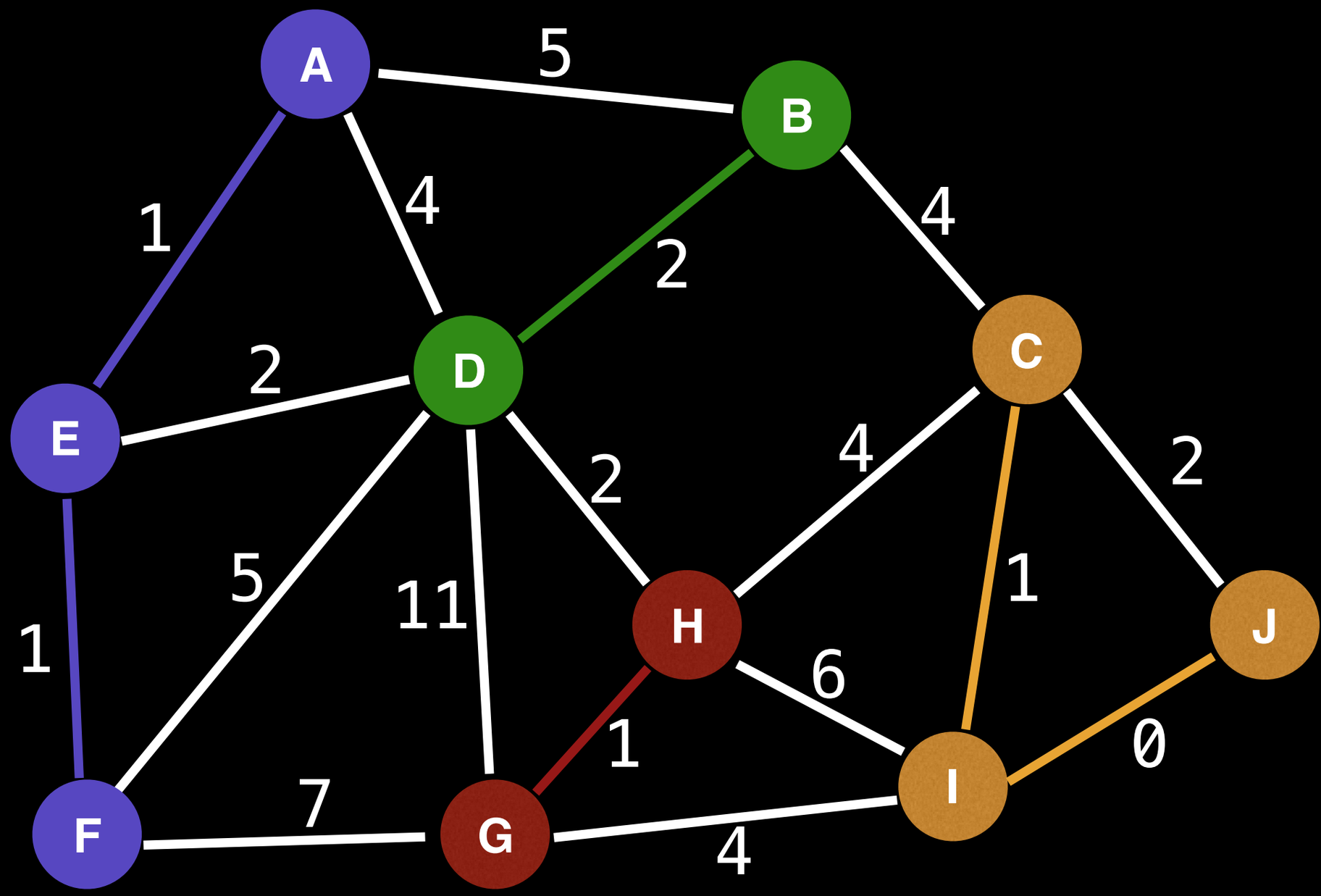
Union Find application: Kruskal's Minimum Spanning Tree

I to J = 0
A to E = 1
C to I = 1
E to F = 1
G to H = 1
B to D = 2
C to J = 2
D to E = 2
D to H = 2
A to D = 4
B to C = 4
C to H = 4
G to I = 4
A to B = 5
D to F = 5
H to I = 6
F to G = 7
D to G = 11



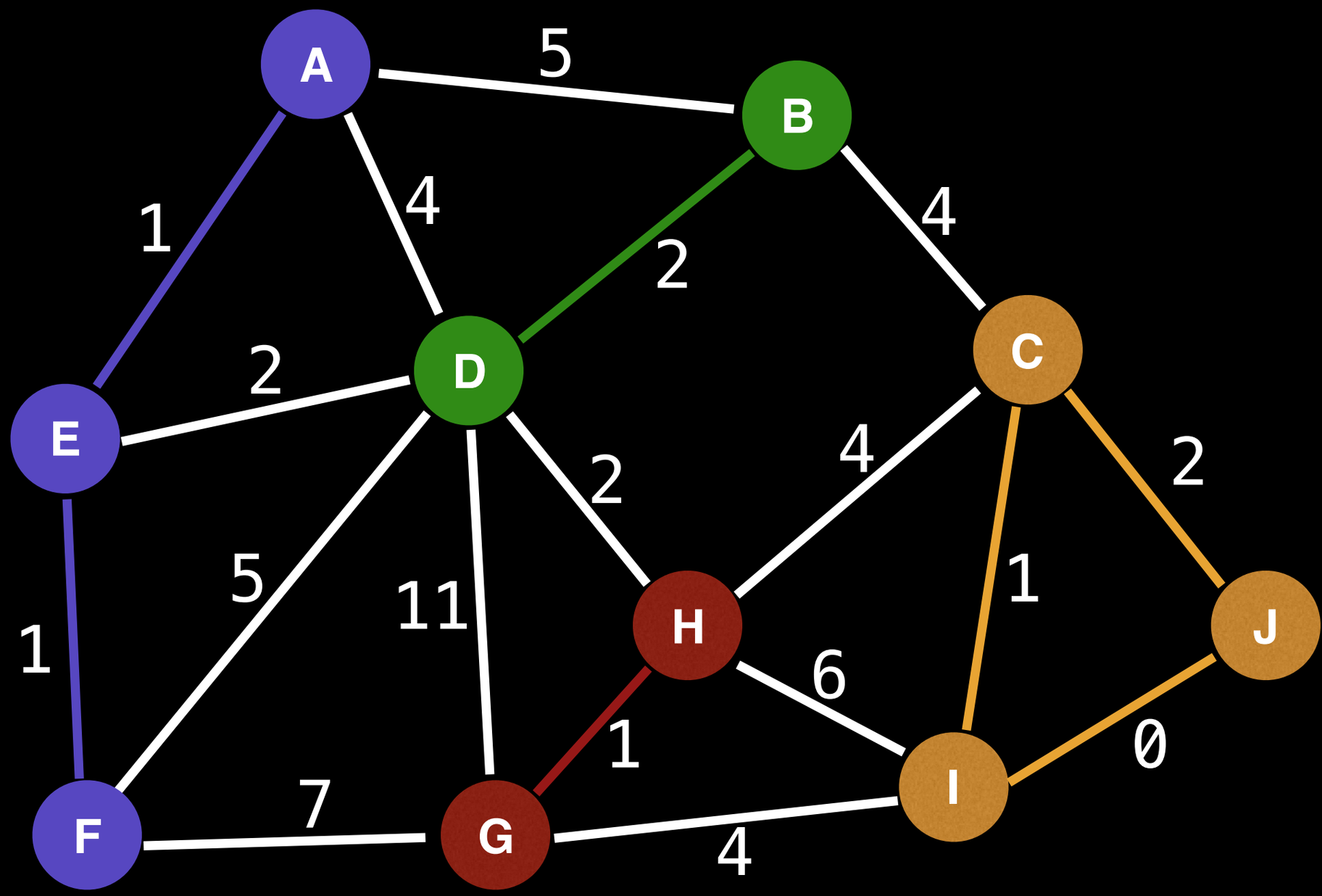
Union Find application: Kruskal's Minimum Spanning Tree

I to J = 0
A to E = 1
C to I = 1
E to F = 1
G to H = 1
B to D = 2
C to J = 2
D to E = 2
D to H = 2
A to D = 4
B to C = 4
C to H = 4
G to I = 4
A to B = 5
D to F = 5
H to I = 6
F to G = 7
D to G = 11



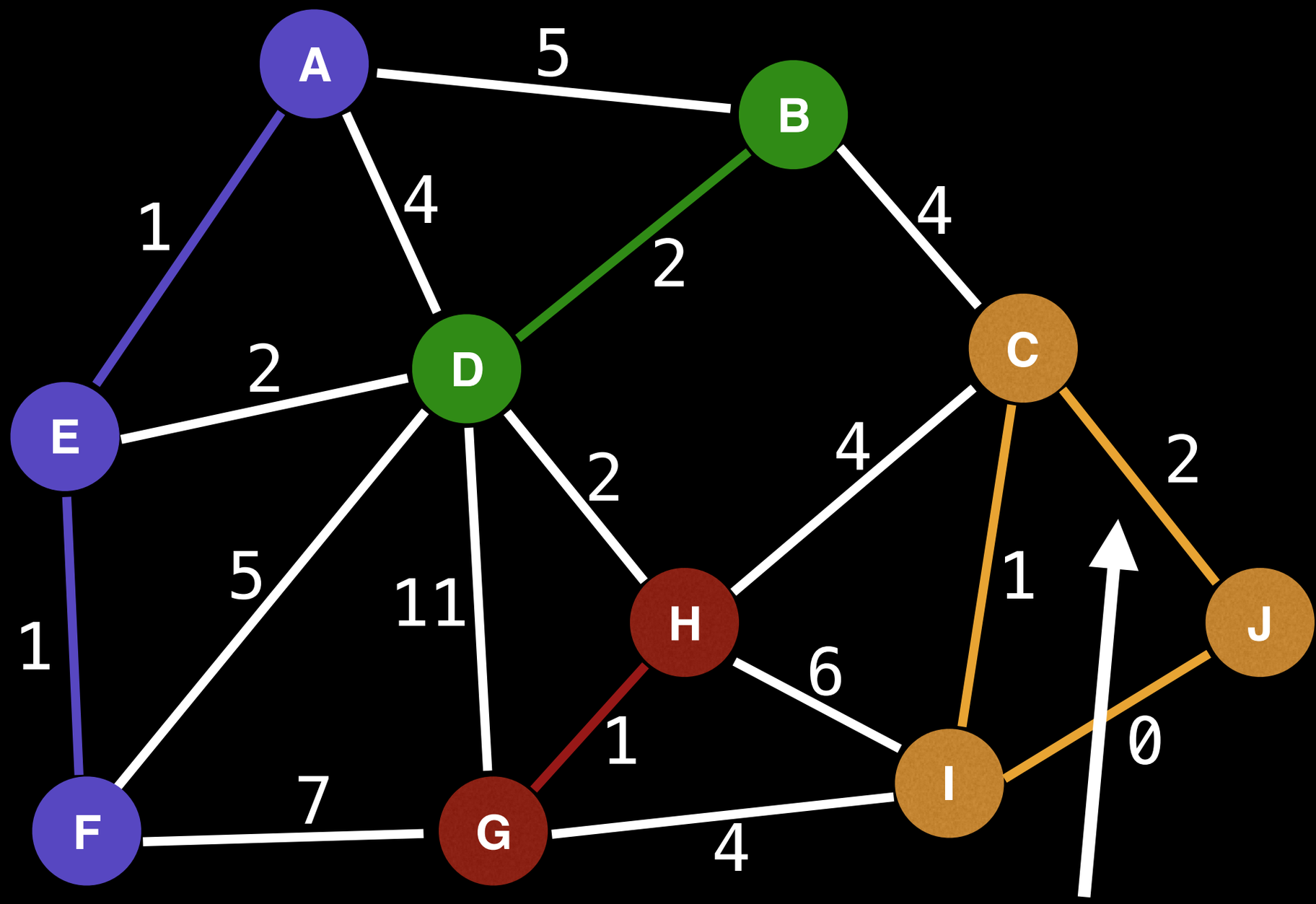
Union Find application: Kruskal's Minimum Spanning Tree

I to J = 0
A to E = 1
C to I = 1
E to F = 1
G to H = 1
B to D = 2
C to J = 2
D to E = 2
D to H = 2
A to D = 4
B to C = 4
C to H = 4
G to I = 4
A to B = 5
D to F = 5
H to I = 6
F to G = 7
D to G = 11



Union Find application: Kruskal's Minimum Spanning Tree

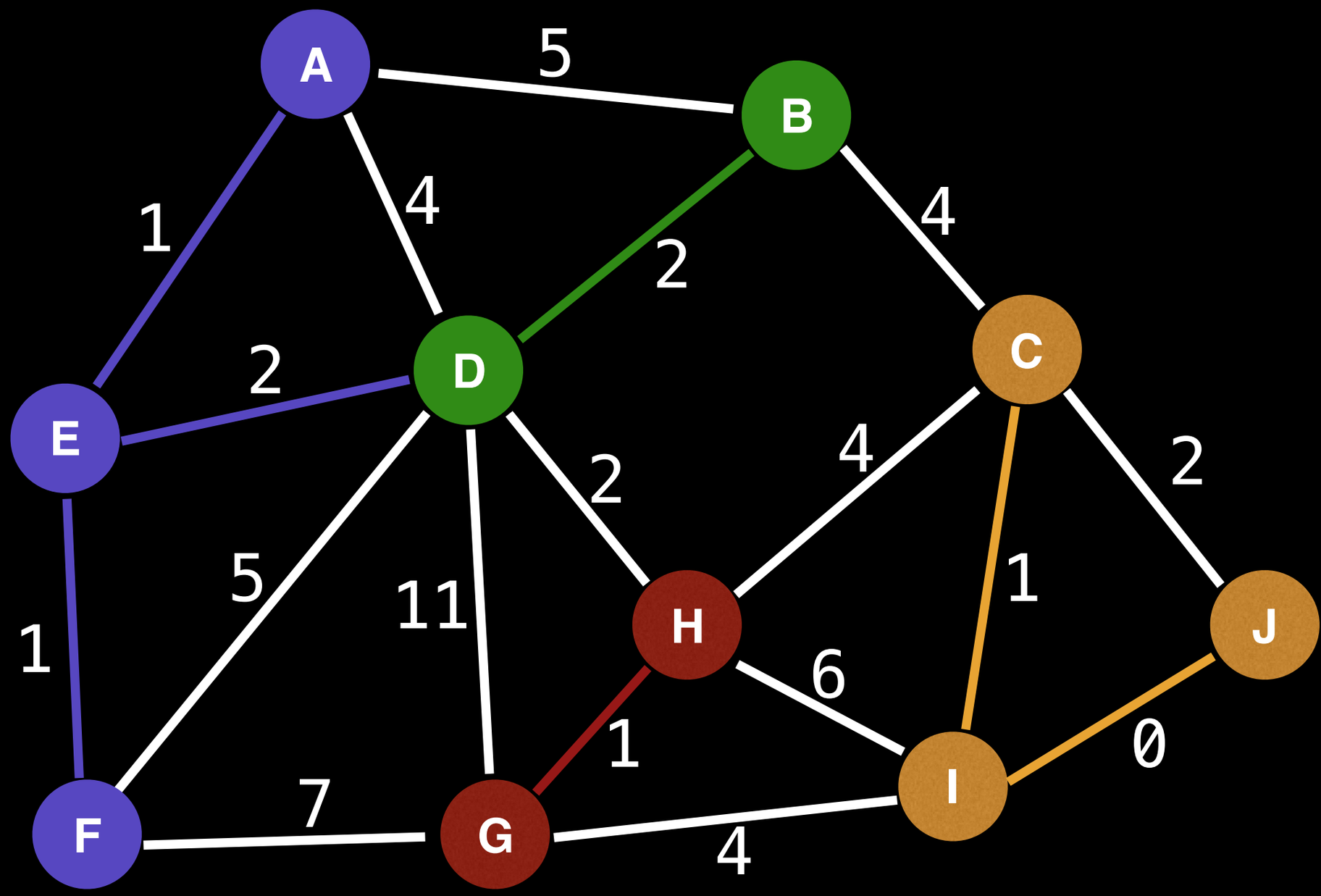
I to J = 0
A to E = 1
C to I = 1
E to F = 1
G to H = 1
B to D = 2
C to J = 2
D to E = 2
D to H = 2
A to D = 4
B to C = 4
C to H = 4
G to I = 4
A to B = 5
D to F = 5
H to I = 6
F to G = 7
D to G = 11



Nodes C,J are already connected in yellow group. This creates a cycle

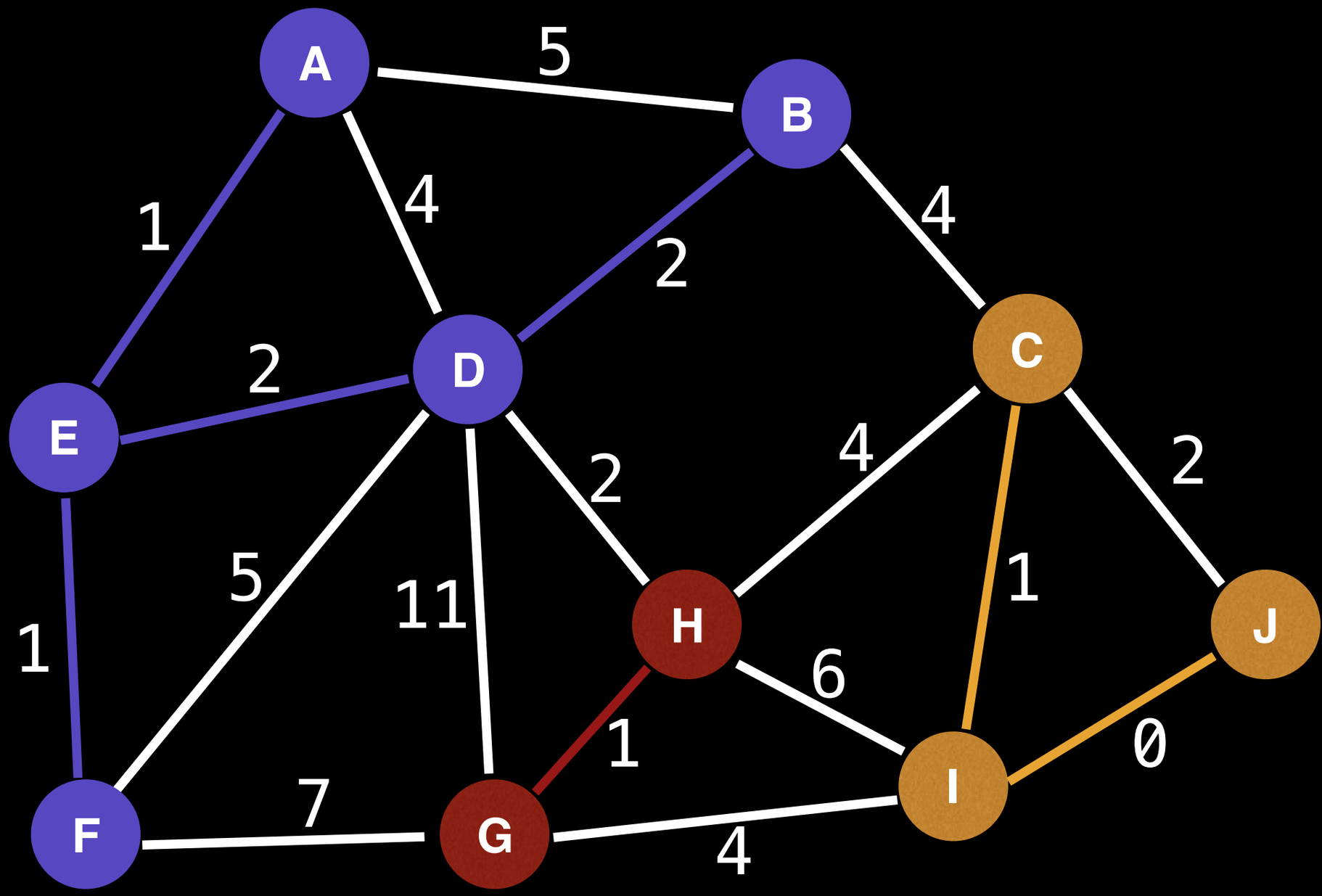
Union Find application: Kruskal's Minimum Spanning Tree

I to J = 0
A to E = 1
C to I = 1
E to F = 1
G to H = 1
B to D = 2
C to J = 2
D to E = 2
D to H = 2
A to D = 4
B to C = 4
C to H = 4
G to I = 4
A to B = 5
D to F = 5
H to I = 6
F to G = 7
D to G = 11



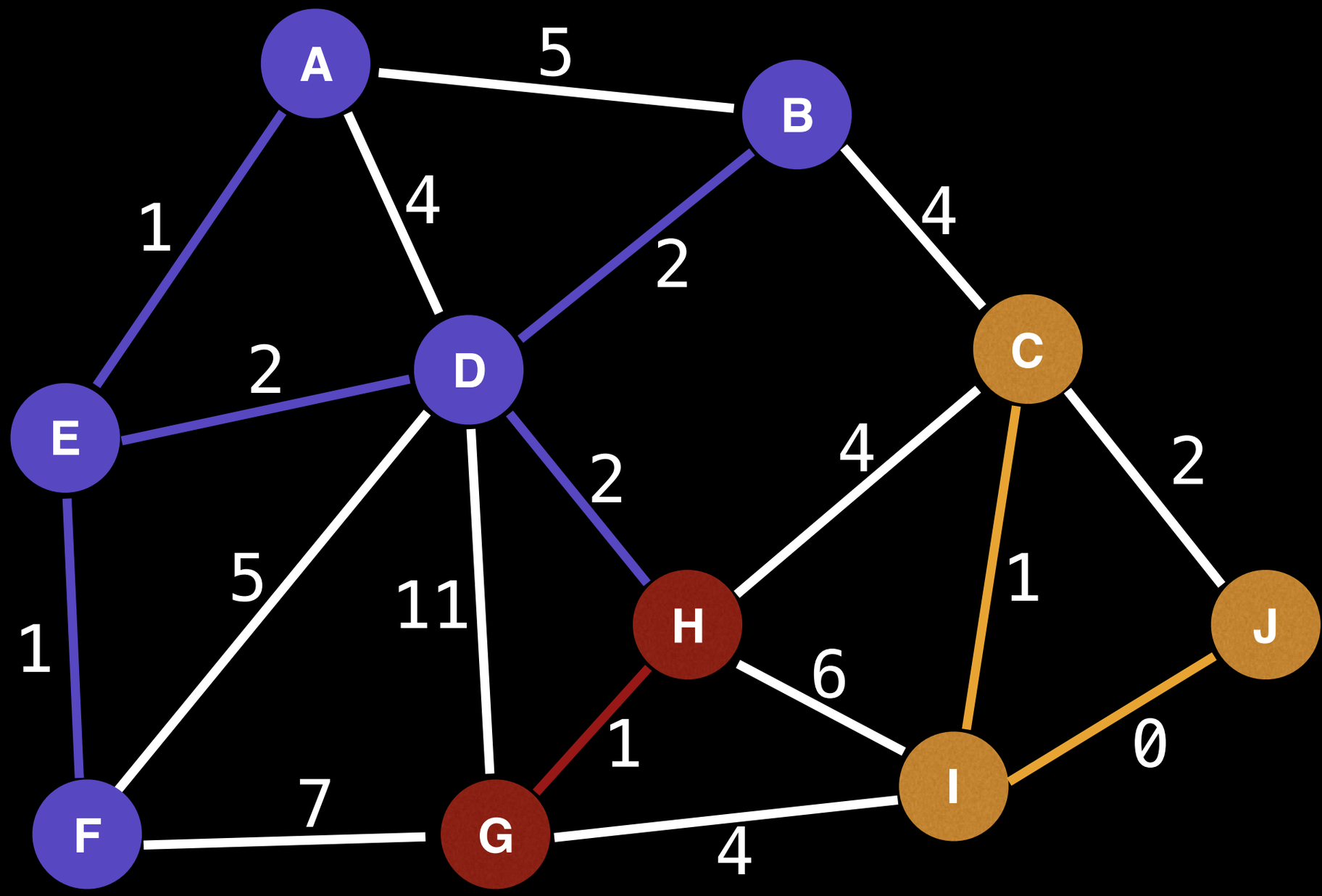
Union Find application: Kruskal's Minimum Spanning Tree

I to J = 0
A to E = 1
C to I = 1
E to F = 1
G to H = 1
B to D = 2
C to J = 2
D to E = 2
D to H = 2
A to D = 4
B to C = 4
C to H = 4
G to I = 4
A to B = 5
D to F = 5
H to I = 6
F to G = 7
D to G = 11



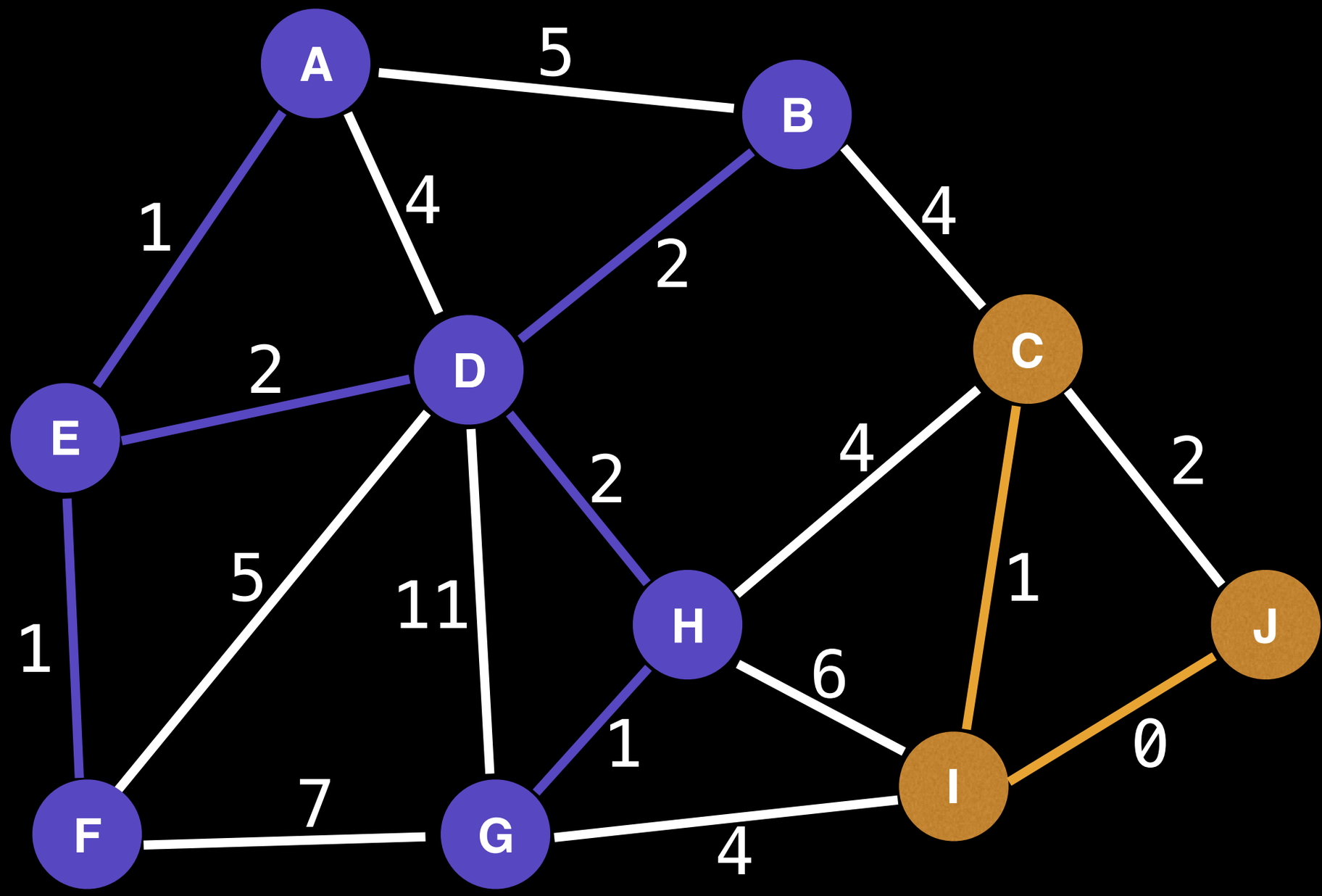
Union Find application: Kruskal's Minimum Spanning Tree

I to J = 0
A to E = 1
C to I = 1
E to F = 1
G to H = 1
B to D = 2
C to J = 2
D to E = 2
D to H = 2
A to D = 4
B to C = 4
C to H = 4
G to I = 4
A to B = 5
D to F = 5
H to I = 6
F to G = 7
D to G = 11



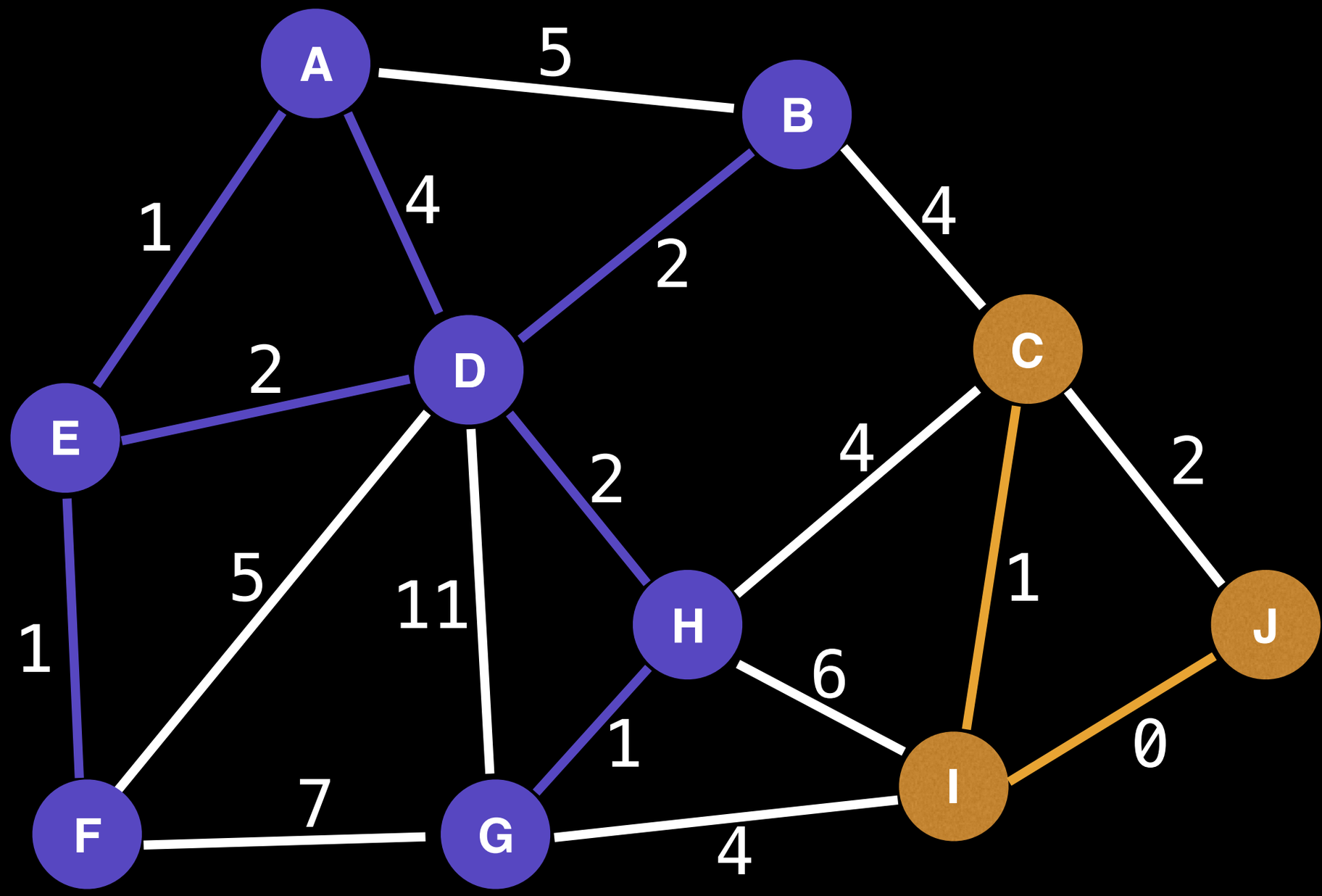
Union Find application: Kruskal's Minimum Spanning Tree

I to J = 0
A to E = 1
C to I = 1
E to F = 1
G to H = 1
B to D = 2
C to J = 2
D to E = 2
D to H = 2
A to D = 4
B to C = 4
C to H = 4
G to I = 4
A to B = 5
D to F = 5
H to I = 6
F to G = 7
D to G = 11



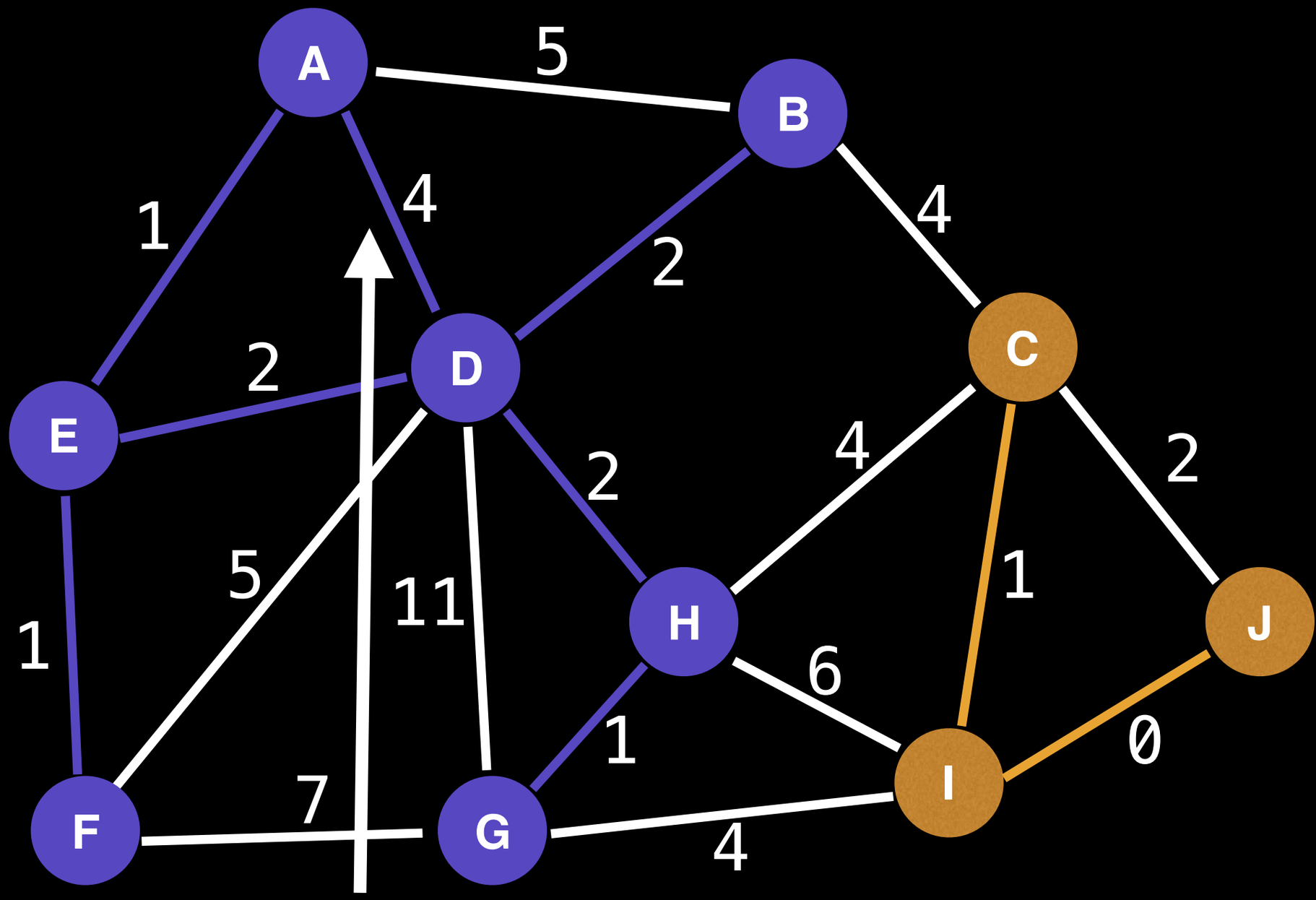
Union Find application: Kruskal's Minimum Spanning Tree

I to J = 0
A to E = 1
C to I = 1
E to F = 1
G to H = 1
B to D = 2
C to J = 2
D to E = 2
D to H = 2
A to D = 4
B to C = 4
C to H = 4
G to I = 4
A to B = 5
D to F = 5
H to I = 6
F to G = 7
D to G = 11



Union Find application: Kruskal's Minimum Spanning Tree

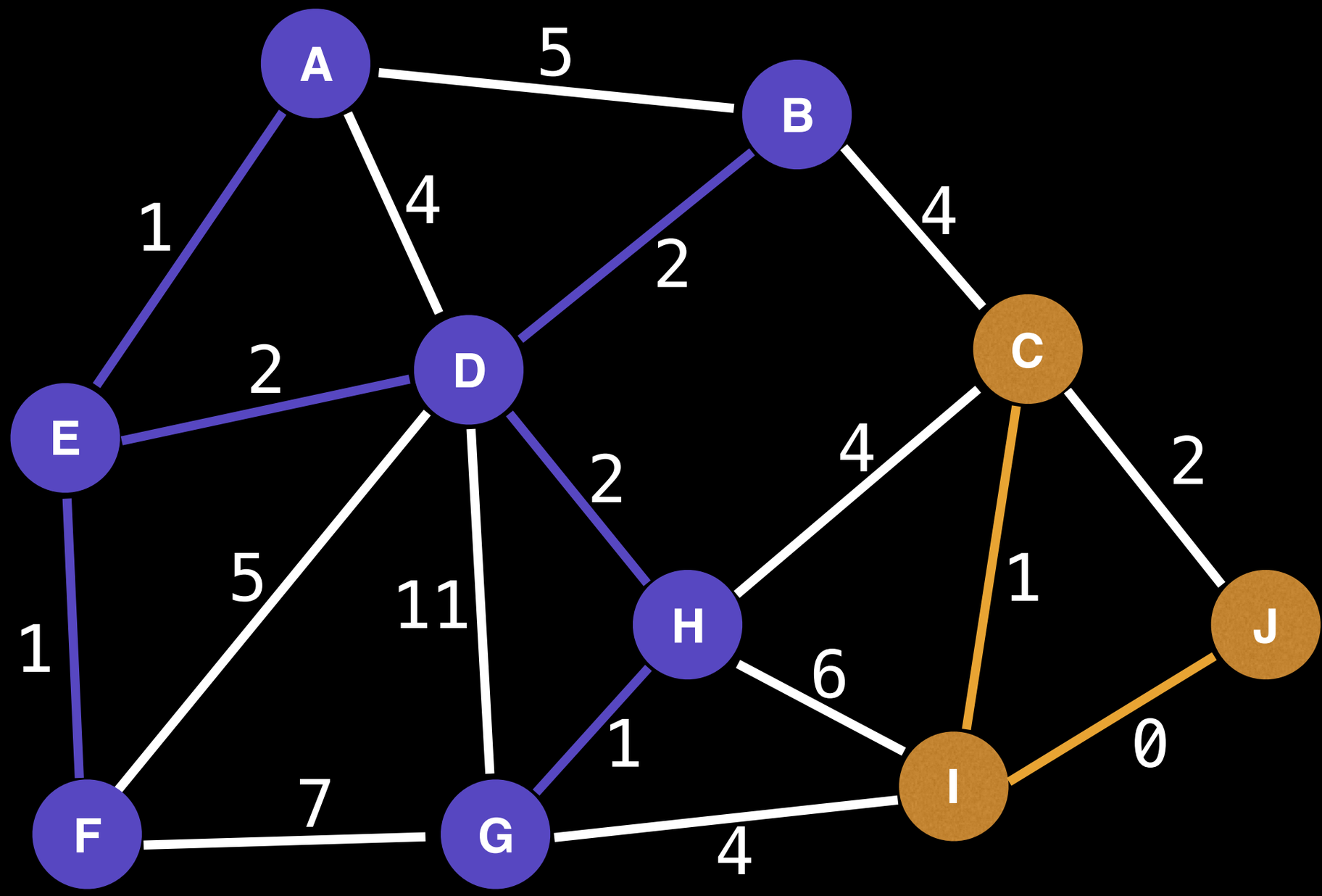
I to J = 0
A to E = 1
C to I = 1
E to F = 1
G to H = 1
B to D = 2
C to J = 2
D to E = 2
D to H = 2
A to D = 4
B to C = 4
C to H = 4
G to I = 4
A to B = 5
D to F = 5
H to I = 6
F to G = 7
D to G = 11



Nodes A,D are already connected in purple group. This creates a cycle

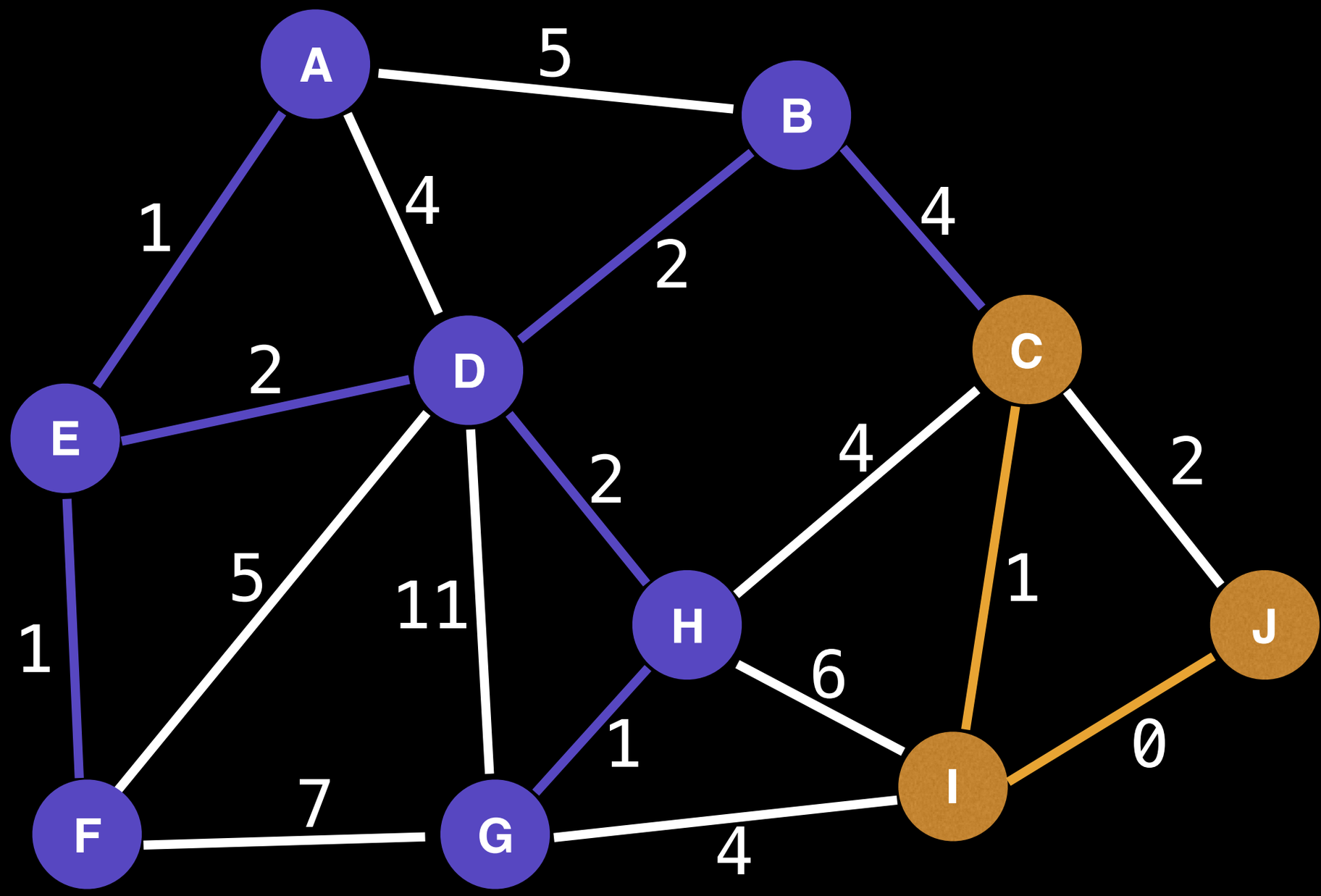
Union Find application: Kruskal's Minimum Spanning Tree

I to J = 0
A to E = 1
C to I = 1
E to F = 1
G to H = 1
B to D = 2
C to J = 2
D to E = 2
D to H = 2
A to D = 4
B to C = 4
C to H = 4
G to I = 4
A to B = 5
D to F = 5
H to I = 6
F to G = 7
D to G = 11



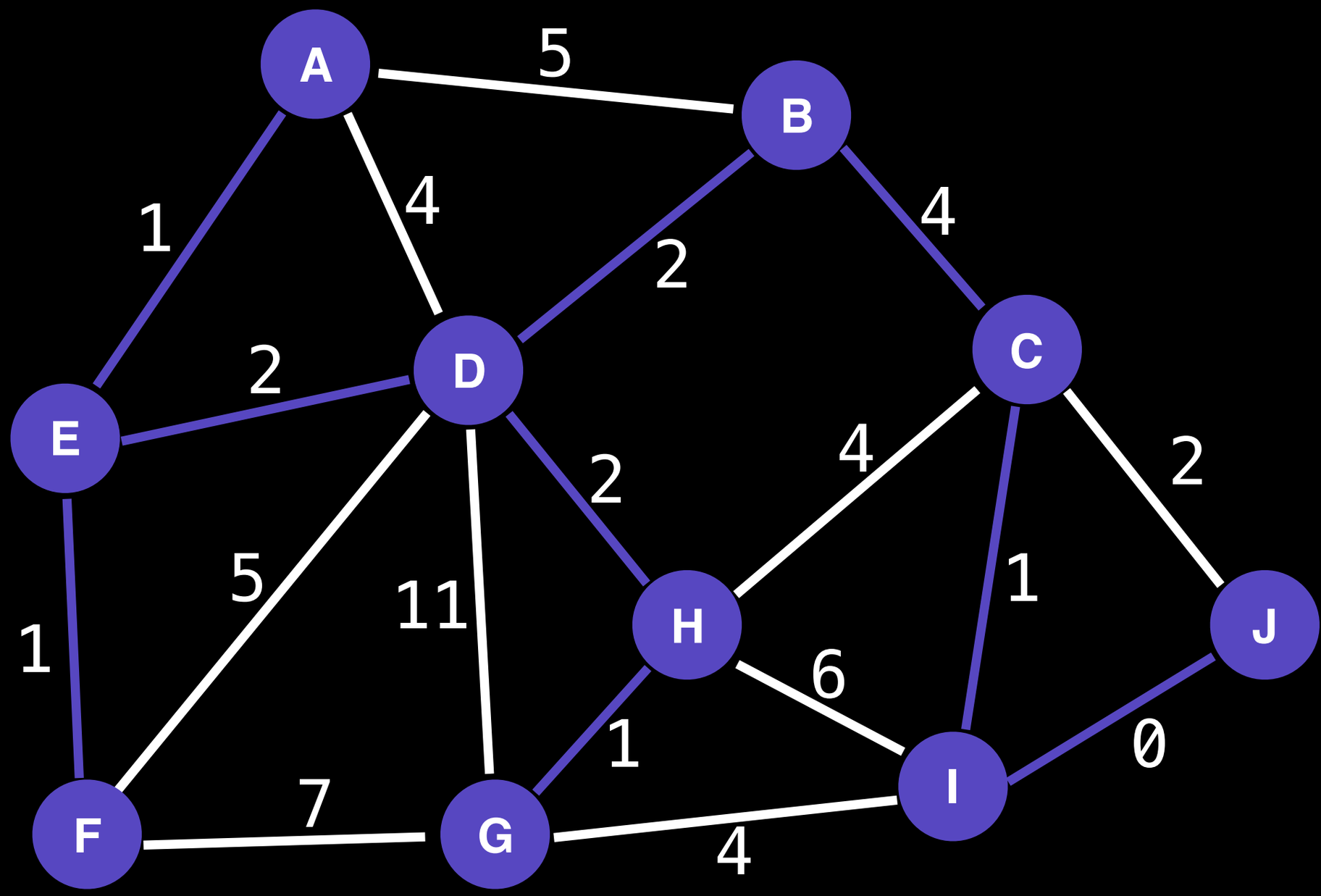
Union Find application: Kruskal's Minimum Spanning Tree

I to J = 0
A to E = 1
C to I = 1
E to F = 1
G to H = 1
B to D = 2
C to J = 2
D to E = 2
D to H = 2
A to D = 4
B to C = 4
C to H = 4
G to I = 4
A to B = 5
D to F = 5
H to I = 6
F to G = 7
D to G = 11



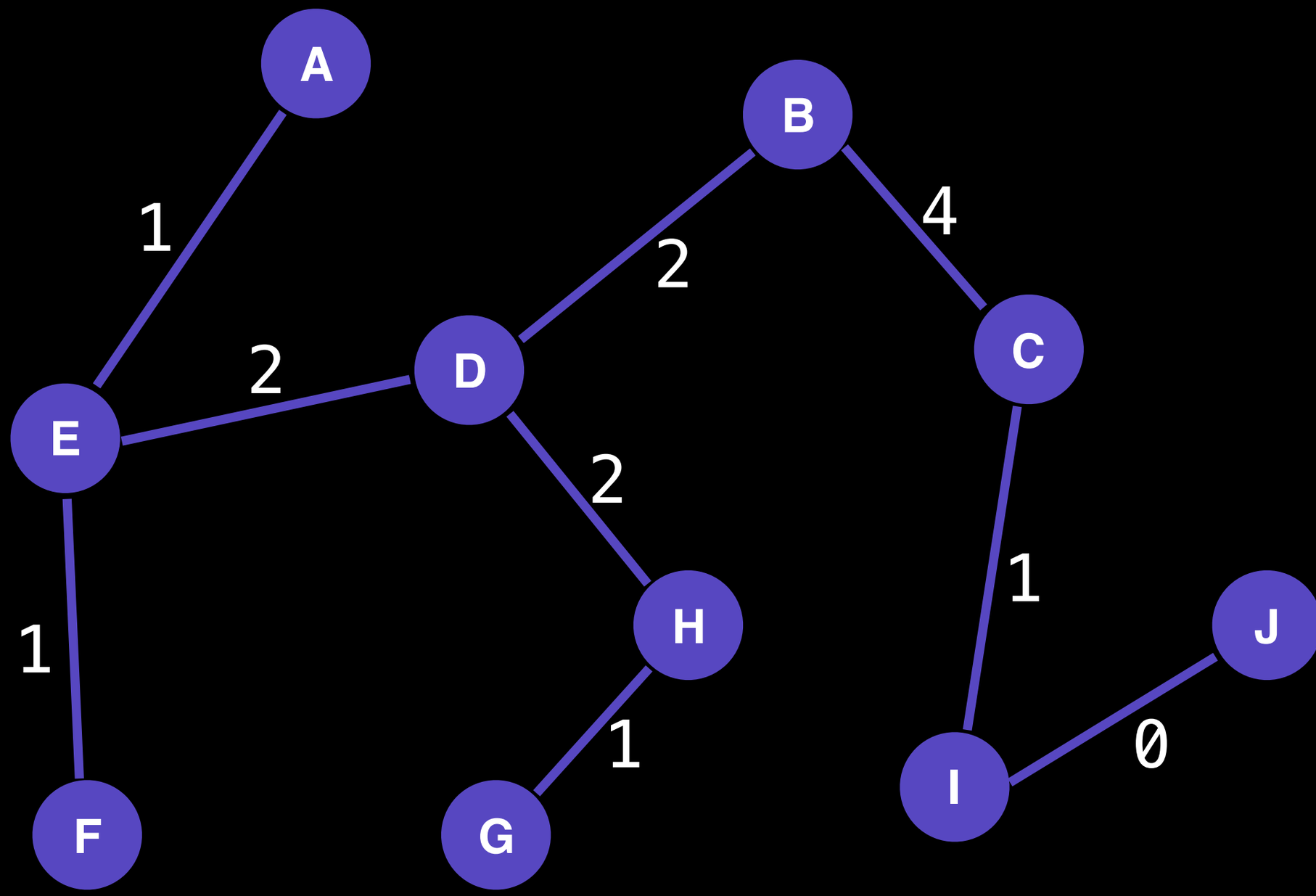
Union Find application: Kruskal's Minimum Spanning Tree

I to J = 0
A to E = 1
C to I = 1
E to F = 1
G to H = 1
B to D = 2
C to J = 2
D to E = 2
D to H = 2
A to D = 4
B to C = 4
C to H = 4
G to I = 4
A to B = 5
D to F = 5
H to I = 6
F to G = 7
D to G = 11



Union Find application: Kruskal's Minimum Spanning Tree

I to J = 0
A to E = 1
C to I = 1
E to F = 1
G to H = 1
B to D = 2
C to J = 2
D to E = 2
D to H = 2
A to D = 4
B to C = 4
C to H = 4
G to I = 4
A to B = 5
D to F = 5
H to I = 6
F to G = 7
D to G = 11



Complexity

Construction	O(n)
Union	α(1)
Find	α(1)
Get component size	α(1)
Check if connected	α(1)
Count components	O(1)

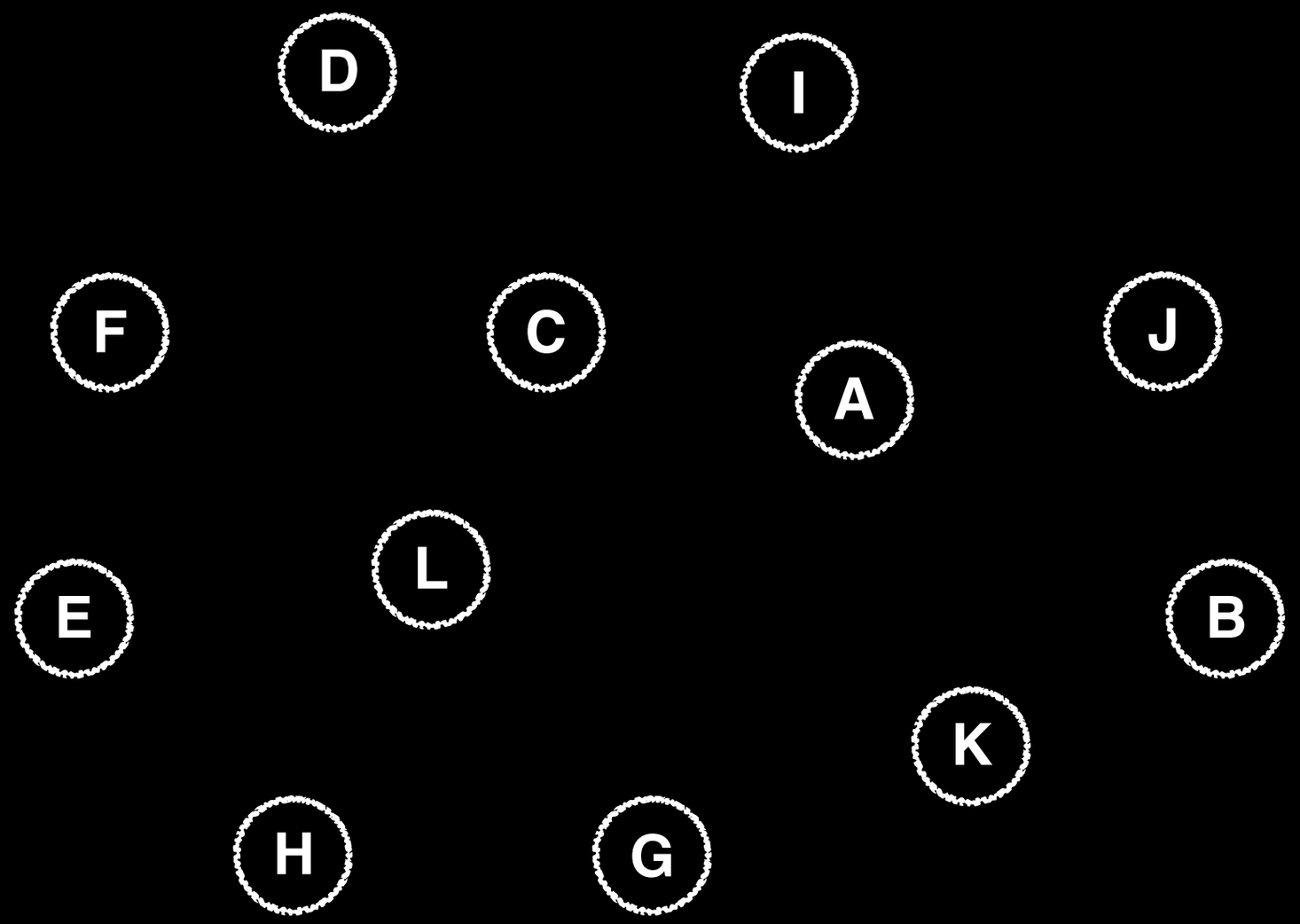
α(1) – Amortized constant time

Union and Find Operations

Union and Find

Instructions:

Union(C,K)
Union(F,E)
Union(A,J)
Union(A,B)
Union(C,D)
Union(D,I)
Union(L,F)
Union(C,A)
Union(A,B)
Union(H,G)
Union(H,F)
Union(H,B)

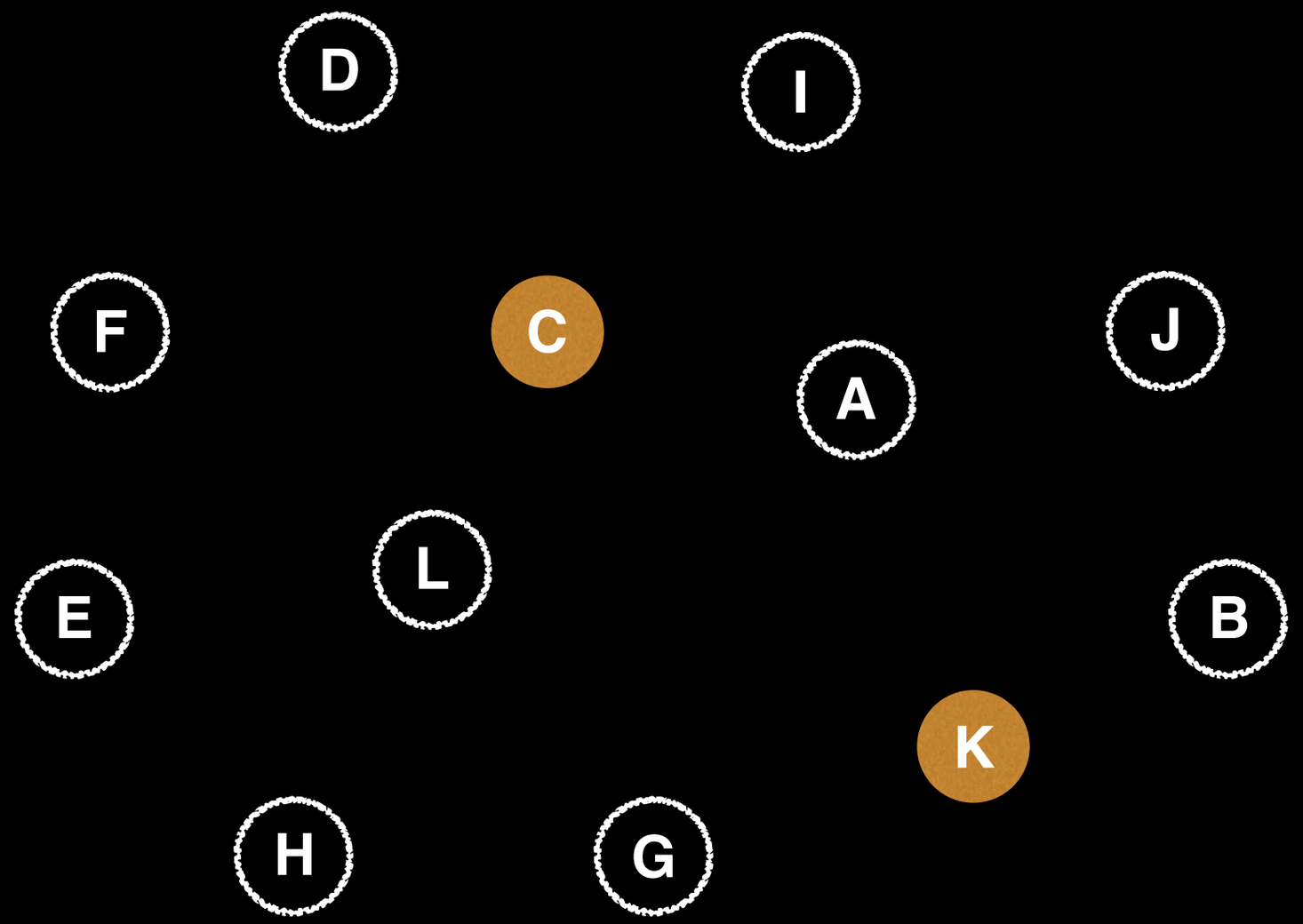


White chalk circles represent groups by themselves. Coloured circles belong to the group with the same colour.

Union and Find

Instructions:

→Union(C,K)
Union(F,E)
Union(A,J)
Union(A,B)
Union(C,D)
Union(D,I)
Union(L,F)
Union(C,A)
Union(A,B)
Union(H,G)
Union(H,F)
Union(H,B)

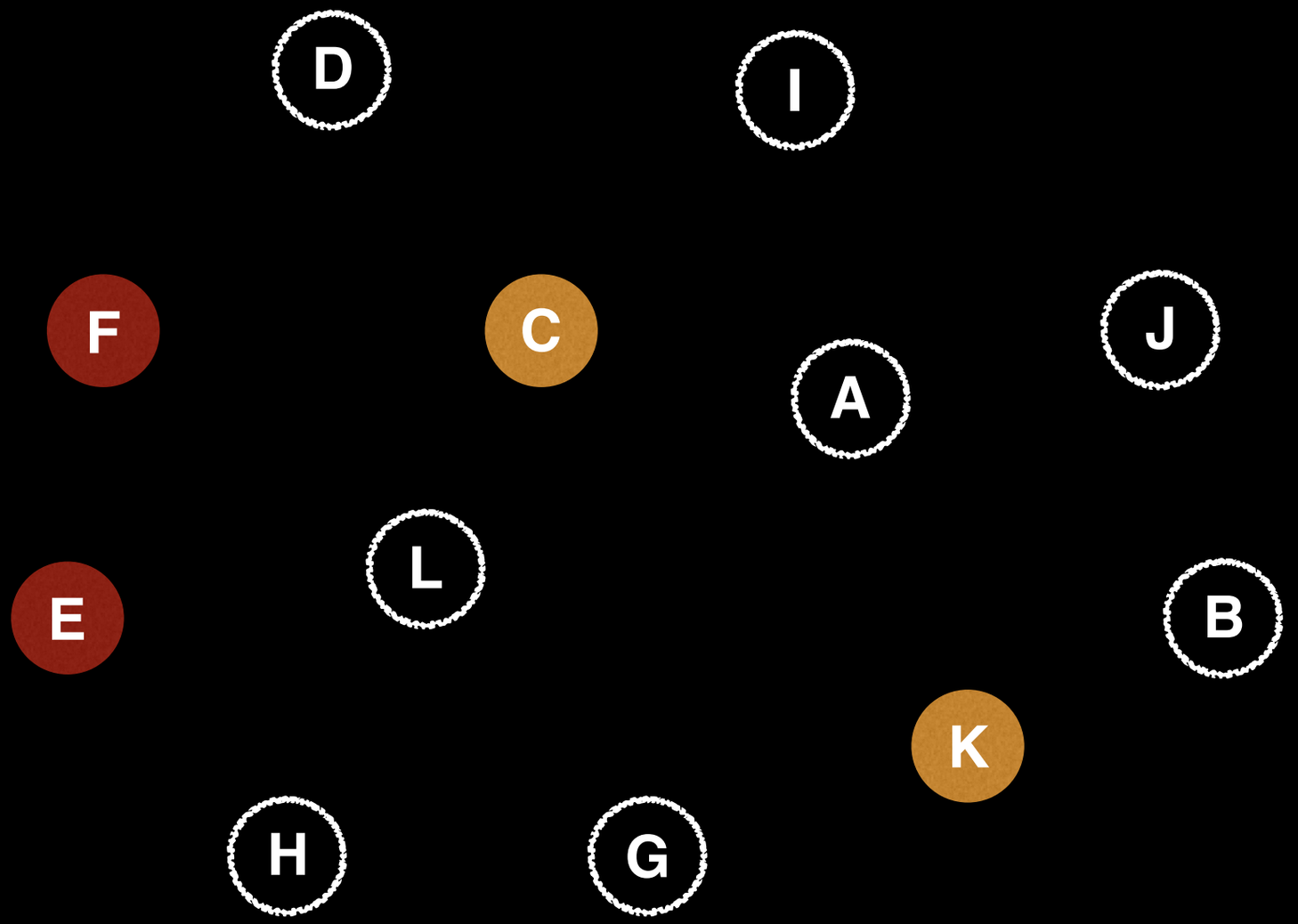


White chalk circles represent groups by themselves. Coloured circles belong to the group with the same colour.

Union and Find

Instructions:

Union(C,K)
→Union(F,E)
Union(A,J)
Union(A,B)
Union(C,D)
Union(D,I)
Union(L,F)
Union(C,A)
Union(A,B)
Union(H,G)
Union(H,F)
Union(H,B)

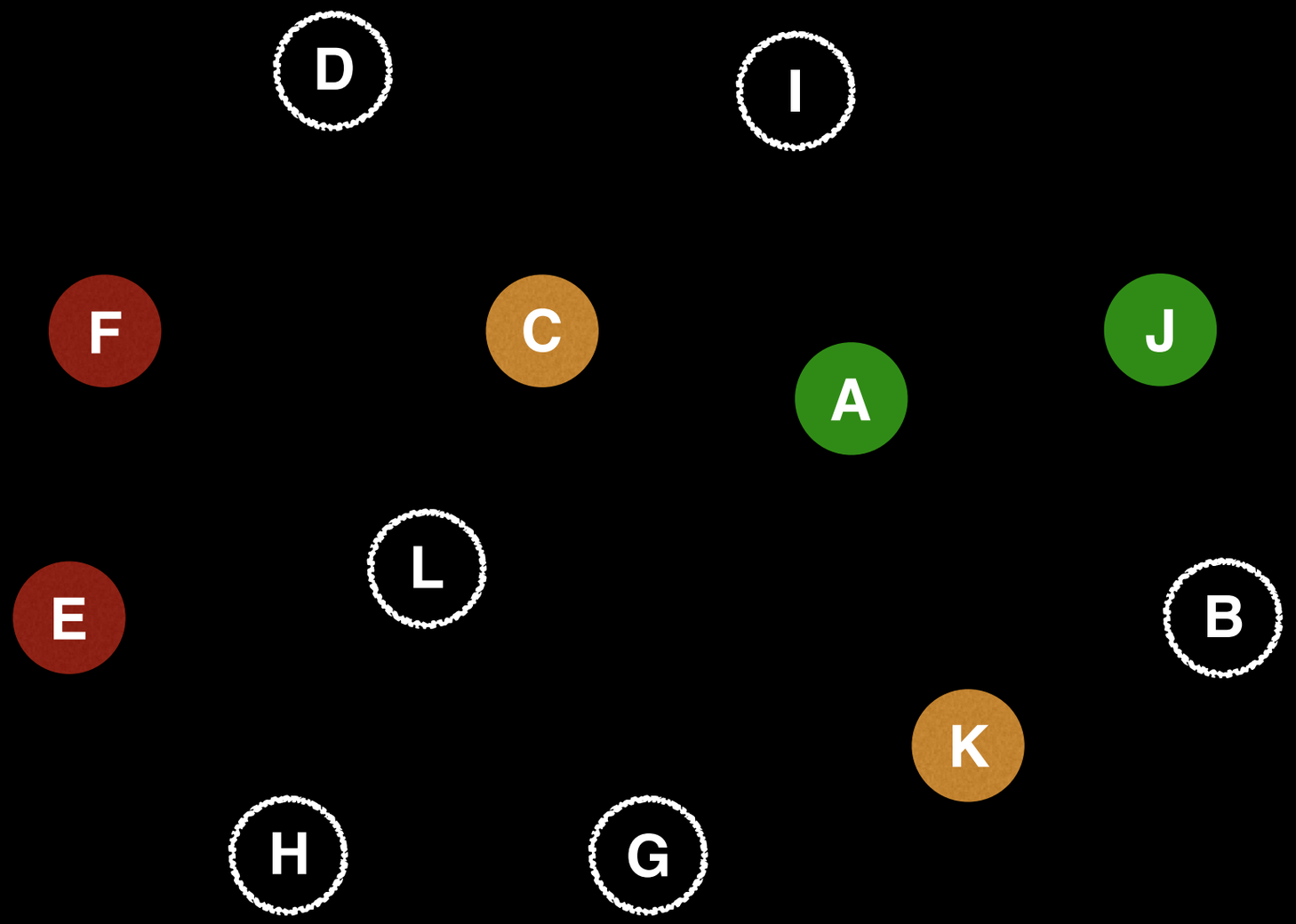


White chalk circles represent groups by themselves. Coloured circles belong to the group with the same colour.

Union and Find

Instructions:

Union(C,K)
Union(F,E)
→Union(A,J)
Union(A,B)
Union(C,D)
Union(D,I)
Union(L,F)
Union(C,A)
Union(A,B)
Union(H,G)
Union(H,F)
Union(H,B)

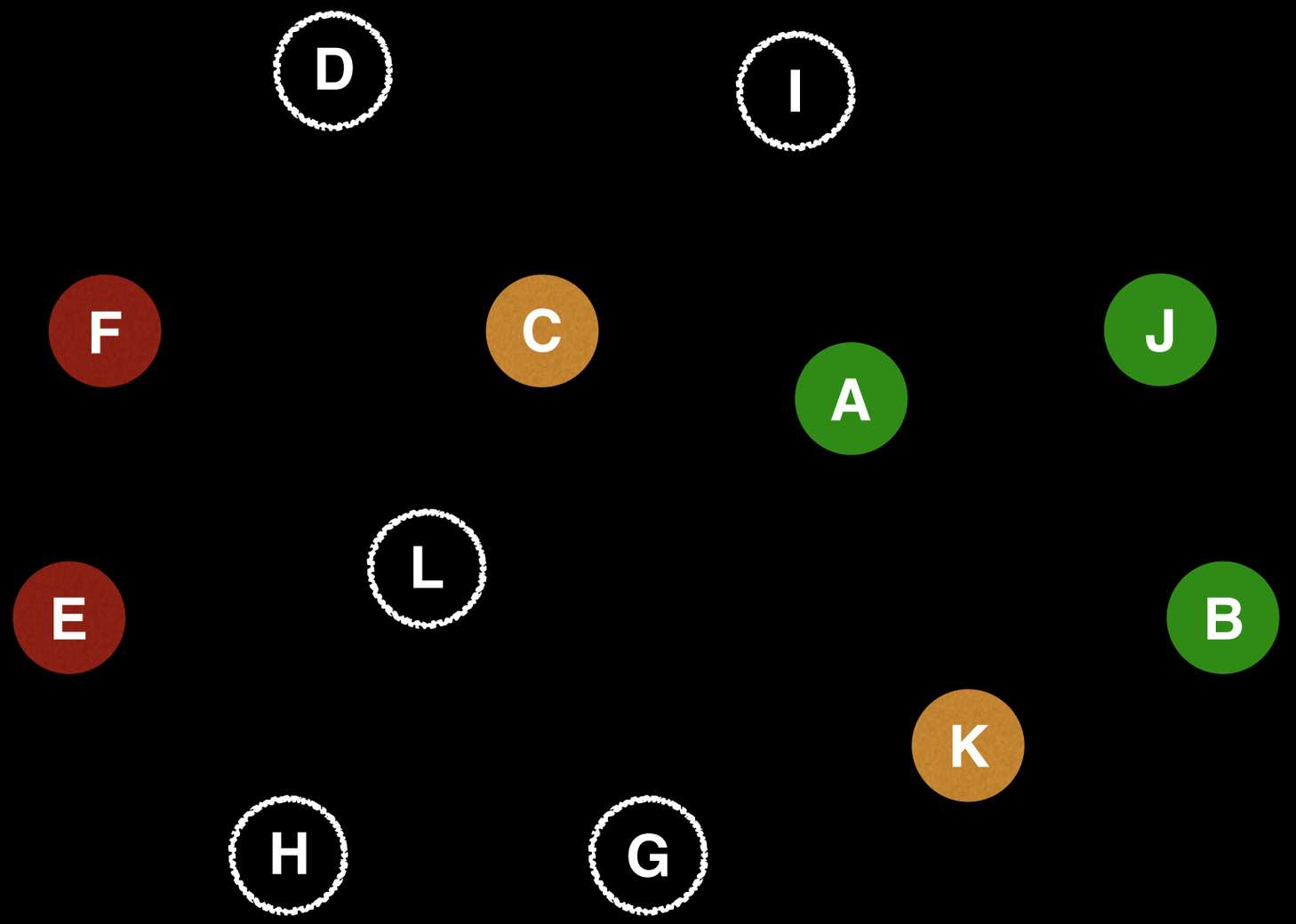


White chalk circles represent groups by themselves. Coloured circles belong to the group with the same colour.

Union and Find

Instructions:

Union(C,K)
Union(F,E)
Union(A,J)
→Union(A,B)
Union(C,D)
Union(D,I)
Union(L,F)
Union(C,A)
Union(A,B)
Union(H,G)
Union(H,F)
Union(H,B)

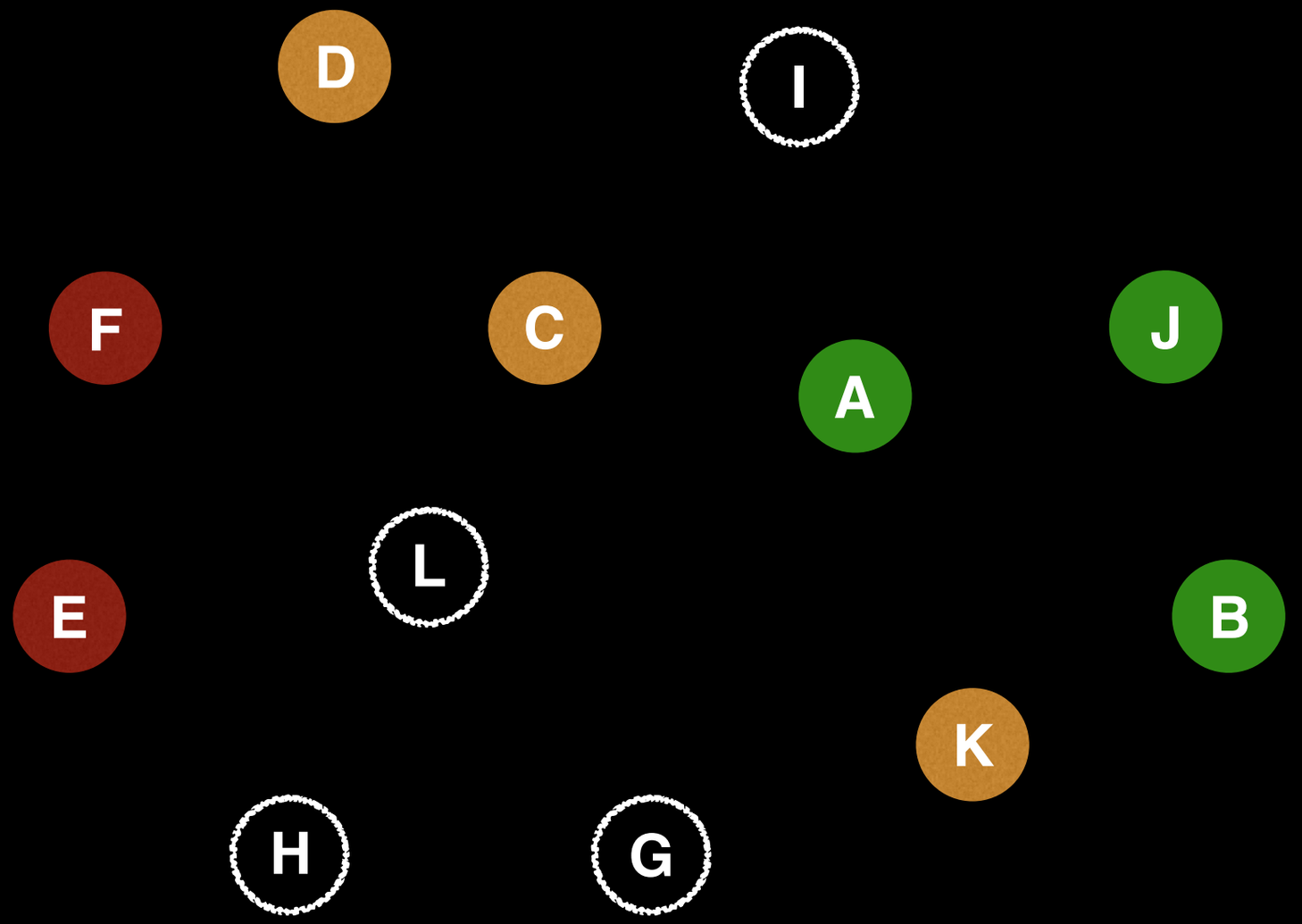


White chalk circles represent groups by themselves. Coloured circles belong to the group with the same colour.

Union and Find

Instructions:

Union(C,K)
Union(F,E)
Union(A,J)
Union(A,B)
→Union(C,D)
Union(D,I)
Union(L,F)
Union(C,A)
Union(A,B)
Union(H,G)
Union(H,F)
Union(H,B)

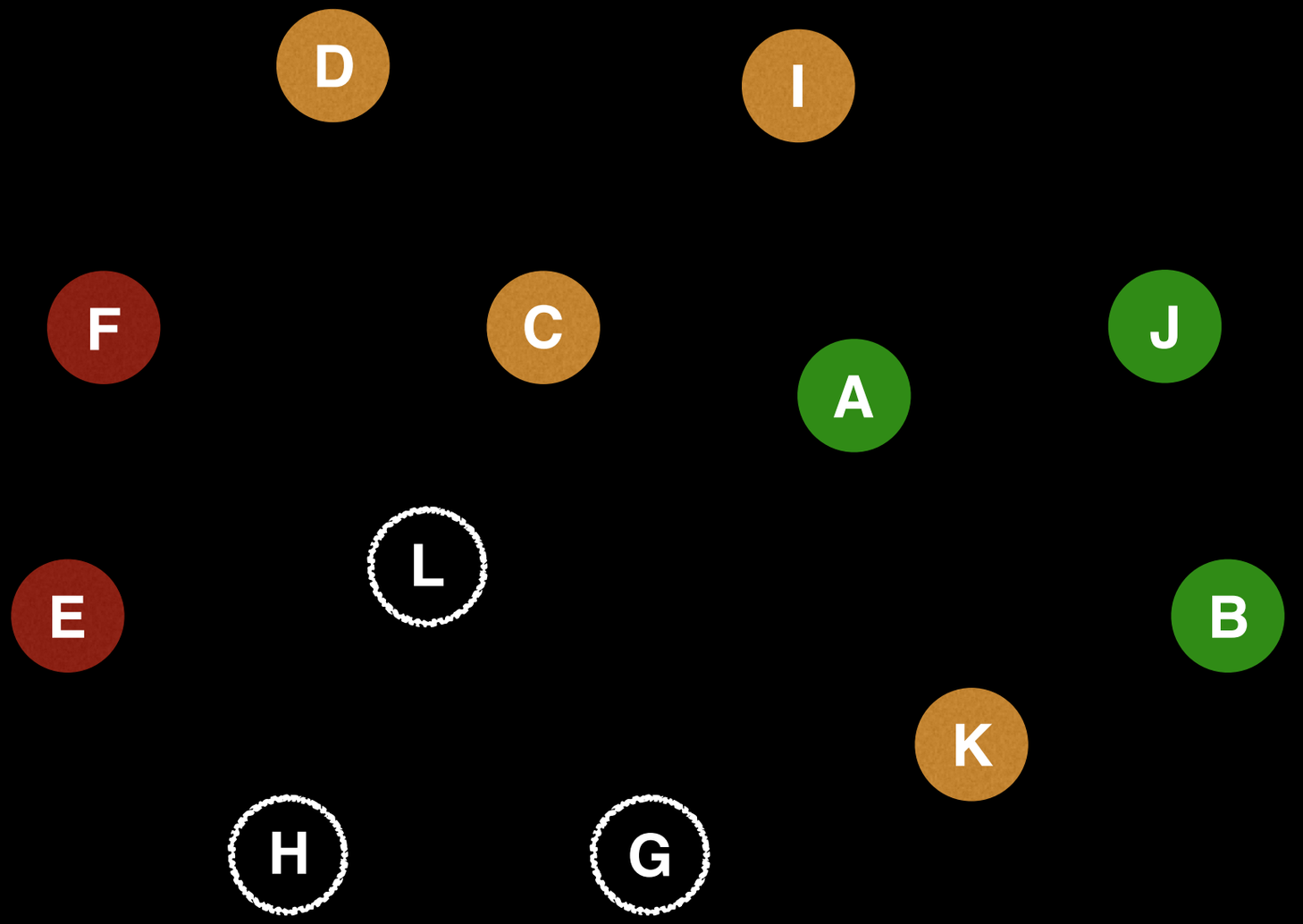


White chalk circles represent groups by themselves. Coloured circles belong to the group with the same colour.

Union and Find

Instructions:

Union(C,K)
Union(F,E)
Union(A,J)
Union(A,B)
Union(C,D)
→Union(D,I)
Union(L,F)
Union(C,A)
Union(A,B)
Union(H,G)
Union(H,F)
Union(H,B)

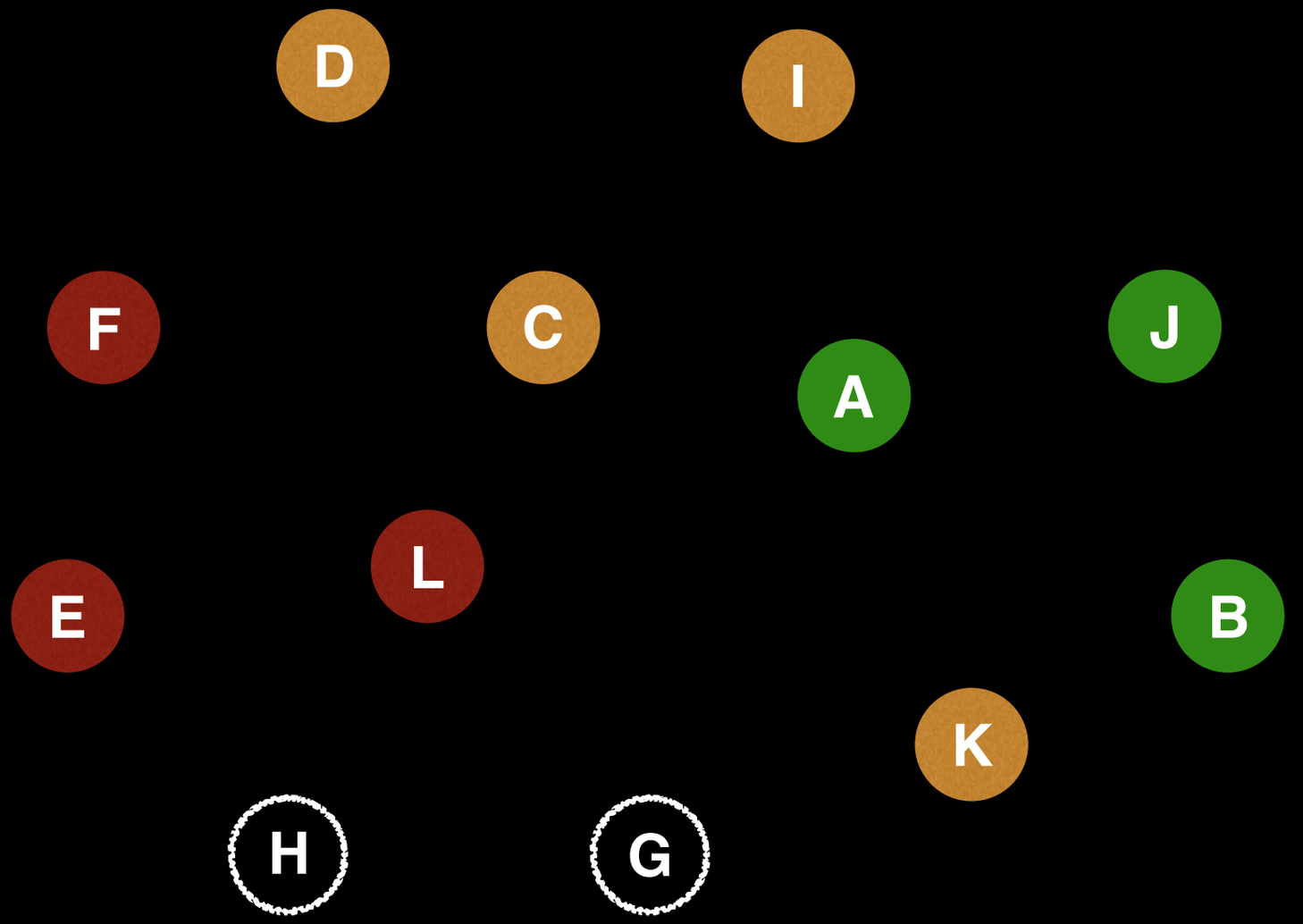


White chalk circles represent groups by themselves. Coloured circles belong to the group with the same colour.

Union and Find

Instructions:

Union(C,K)
Union(F,E)
Union(A,J)
Union(A,B)
Union(C,D)
Union(D,I)
→Union(L,F)
Union(C,A)
Union(A,B)
Union(H,G)
Union(H,F)
Union(H,B)

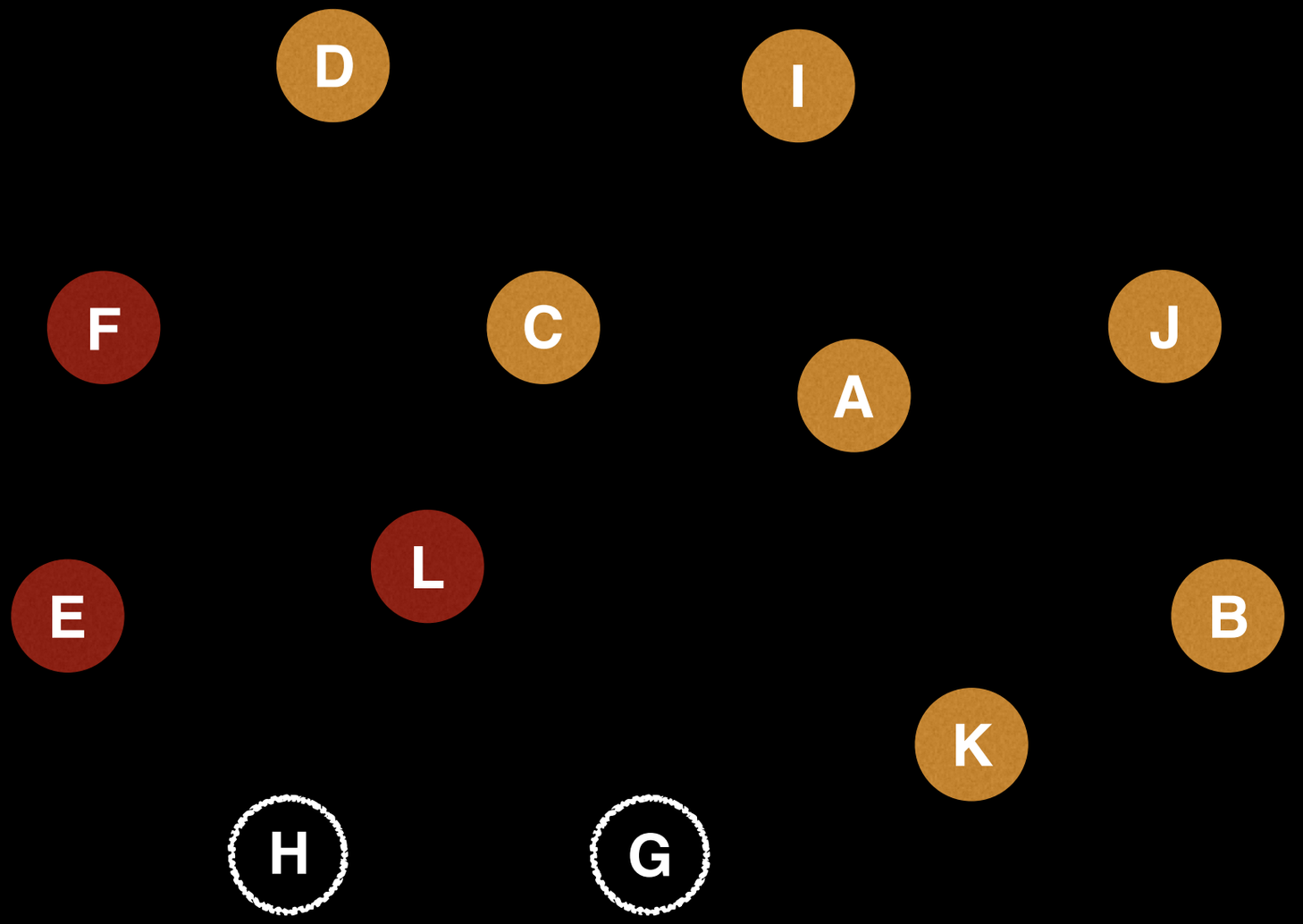


White chalk circles represent groups by themselves. Coloured circles belong to the group with the same colour.

Union and Find

Instructions:

Union(C,K)
Union(F,E)
Union(A,J)
Union(A,B)
Union(C,D)
Union(D,I)
Union(L,F)
→Union(C,A)
Union(A,B)
Union(H,G)
Union(H,F)
Union(H,B)

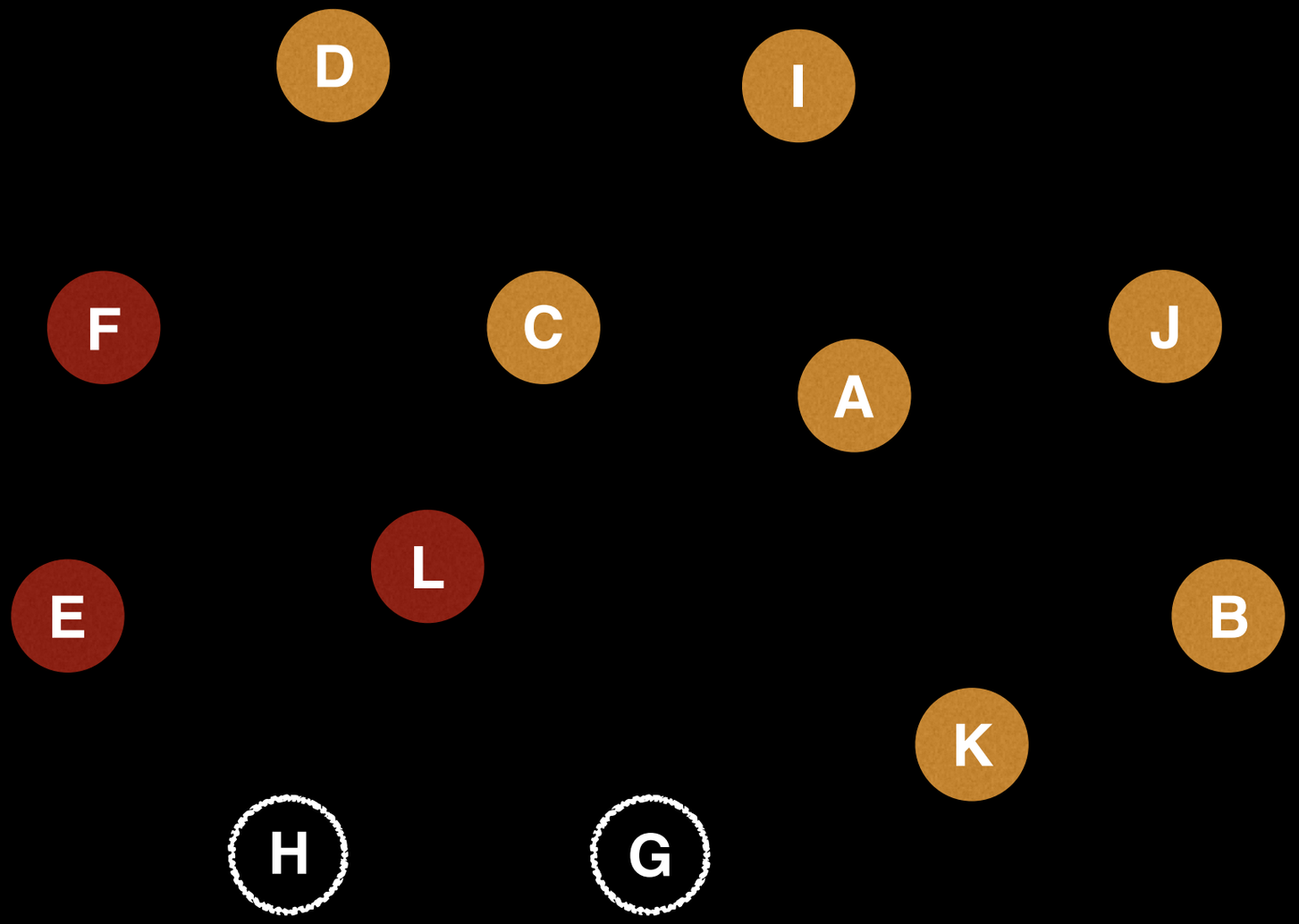


White chalk circles represent groups by themselves. Coloured circles belong to the group with the same colour.

Union and Find

Instructions:

Union(C,K)
Union(F,E)
Union(A,J)
Union(A,B)
Union(C,D)
Union(D,I)
Union(L,F)
Union(C,A)
→Union(A,B)
Union(H,G)
Union(H,F)
Union(H,B)

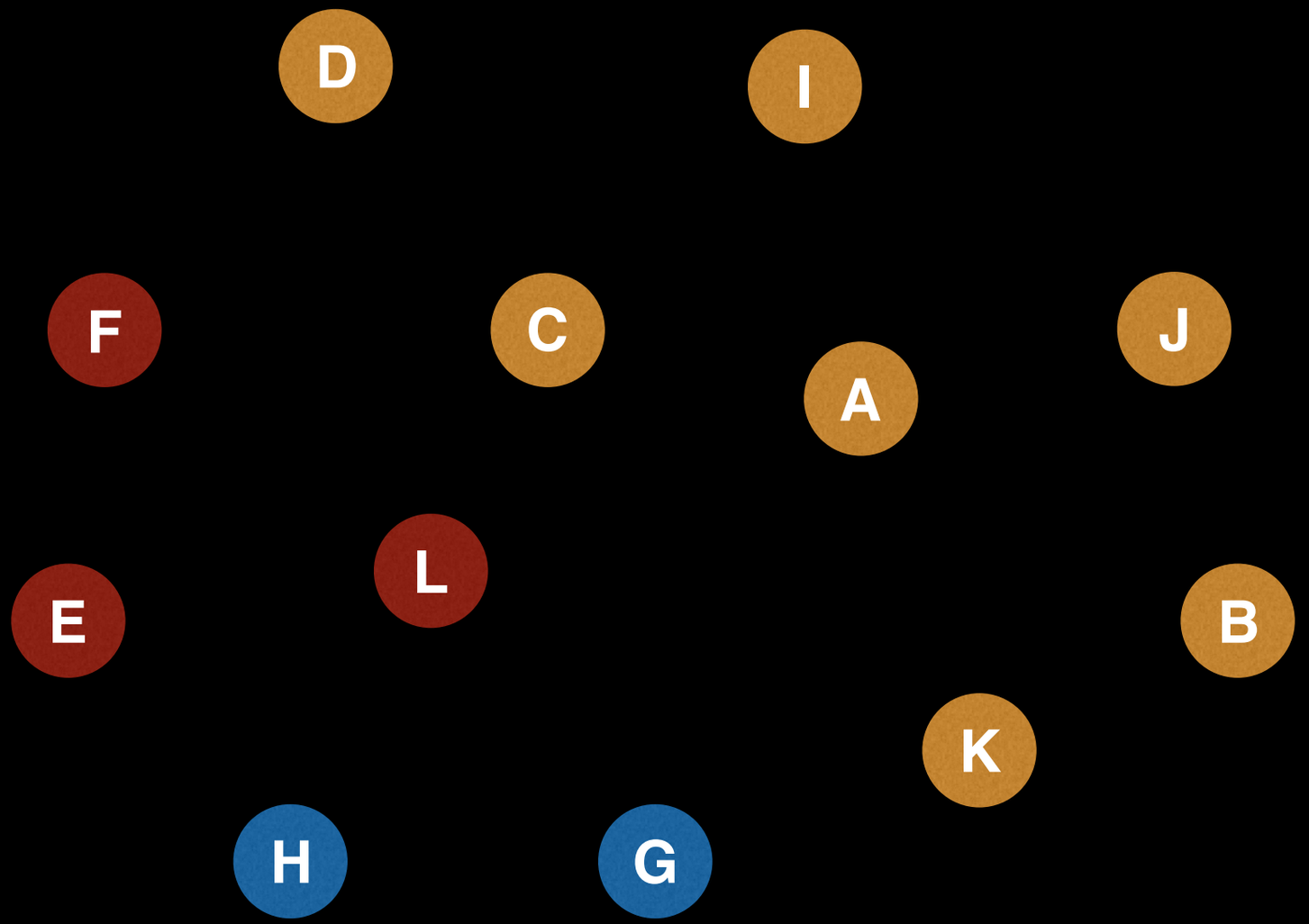


White chalk circles represent groups by themselves. Coloured circles belong to the group with the same colour.

Union and Find

Instructions:

Union(C,K)
Union(F,E)
Union(A,J)
Union(A,B)
Union(C,D)
Union(D,I)
Union(L,F)
Union(C,A)
Union(A,B)
→Union(H,G)
Union(H,F)
Union(H,B)

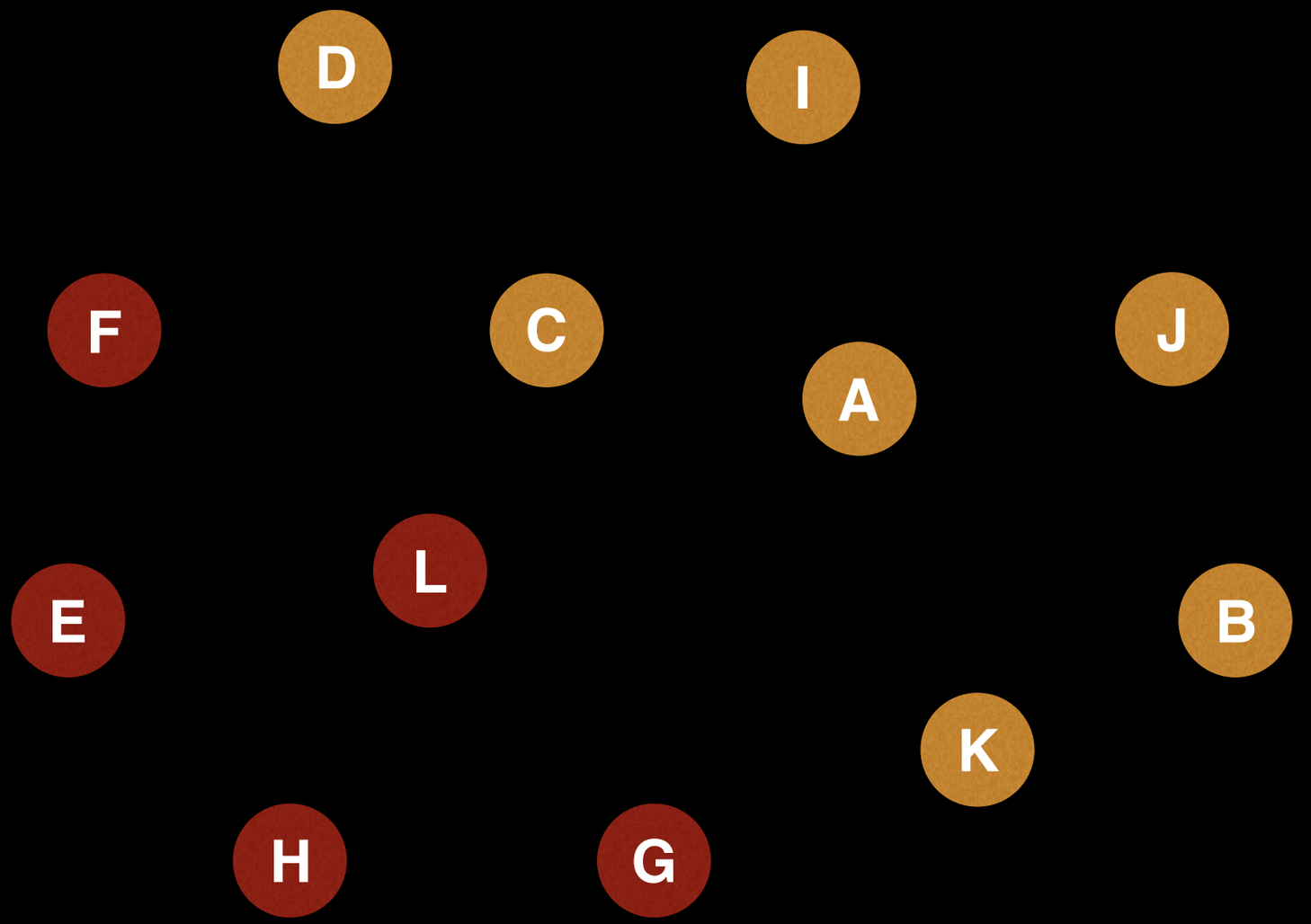


White chalk circles represent groups by themselves. Coloured circles belong to the group with the same colour.

Union and Find

Instructions:

Union(C,K)
Union(F,E)
Union(A,J)
Union(A,B)
Union(C,D)
Union(D,I)
Union(L,F)
Union(C,A)
Union(A,B)
Union(H,G)
→Union(H,F)
Union(H,B)

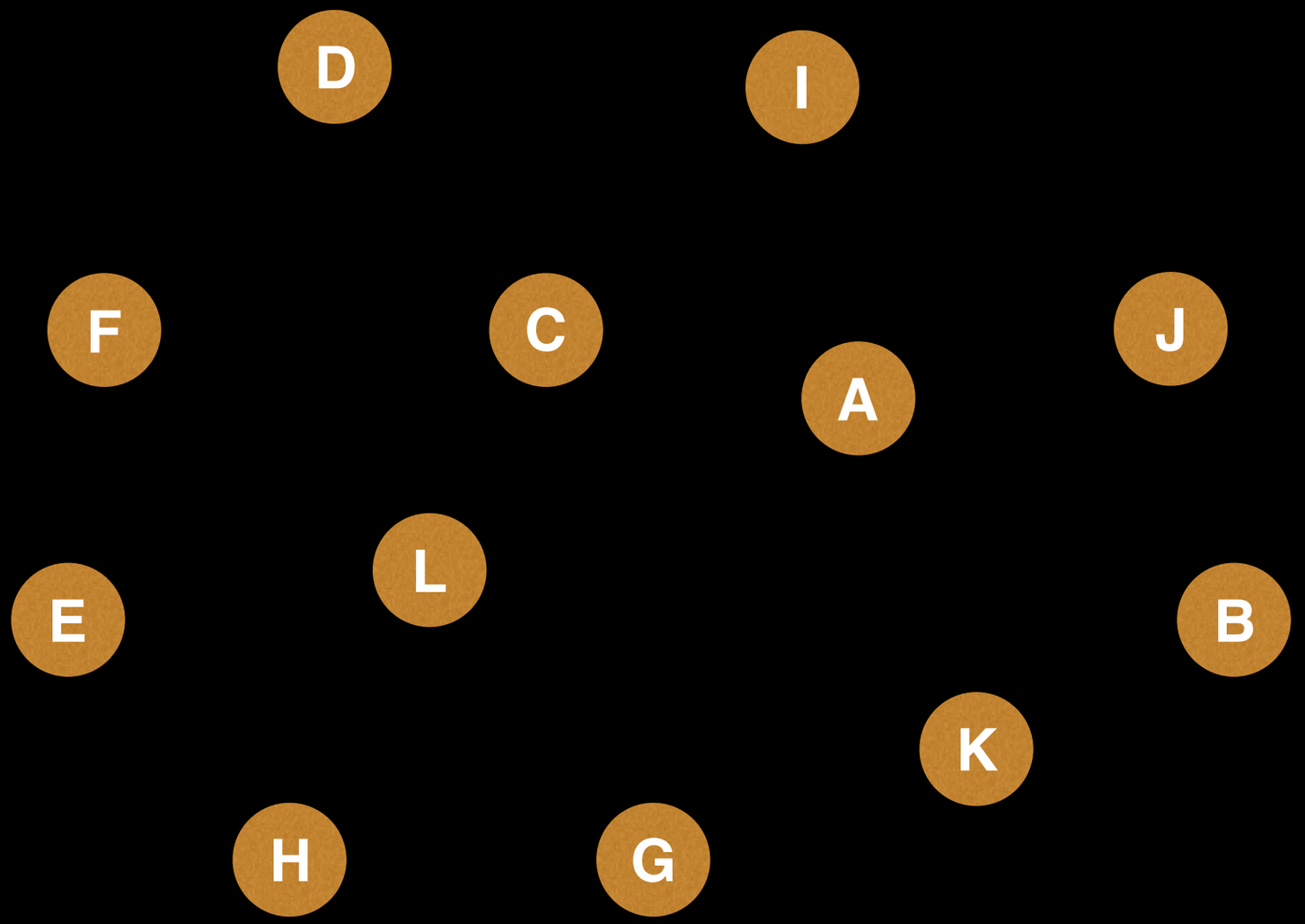


White chalk circles represent groups by themselves. Coloured circles belong to the group with the same colour.

Union and Find

Instructions:

Union(C,K)
Union(F,E)
Union(A,J)
Union(A,B)
Union(C,D)
Union(D,I)
Union(L,F)
Union(C,A)
Union(A,B)
Union(H,G)
Union(H,F)
→Union(H,B)



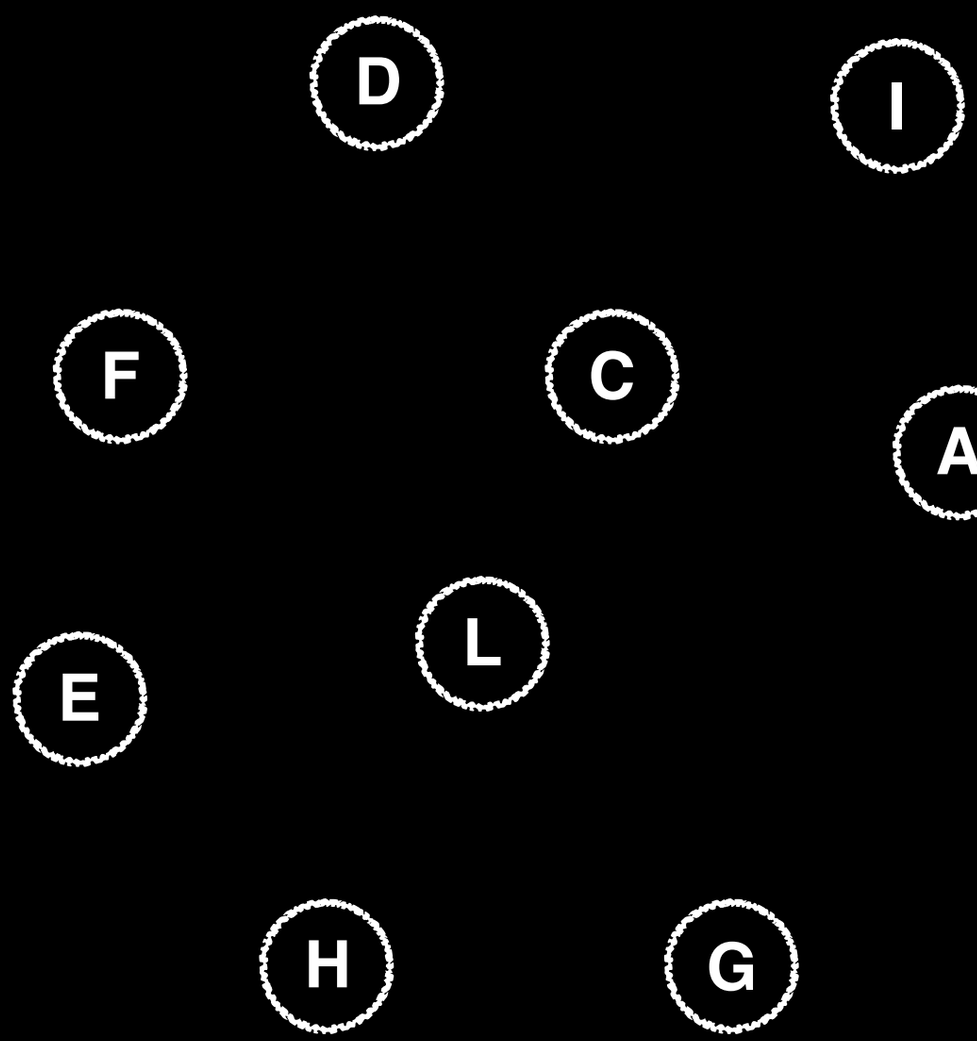
White chalk circles represent groups by themselves. Coloured circles belong to the group with the same colour.

Creating Union Find

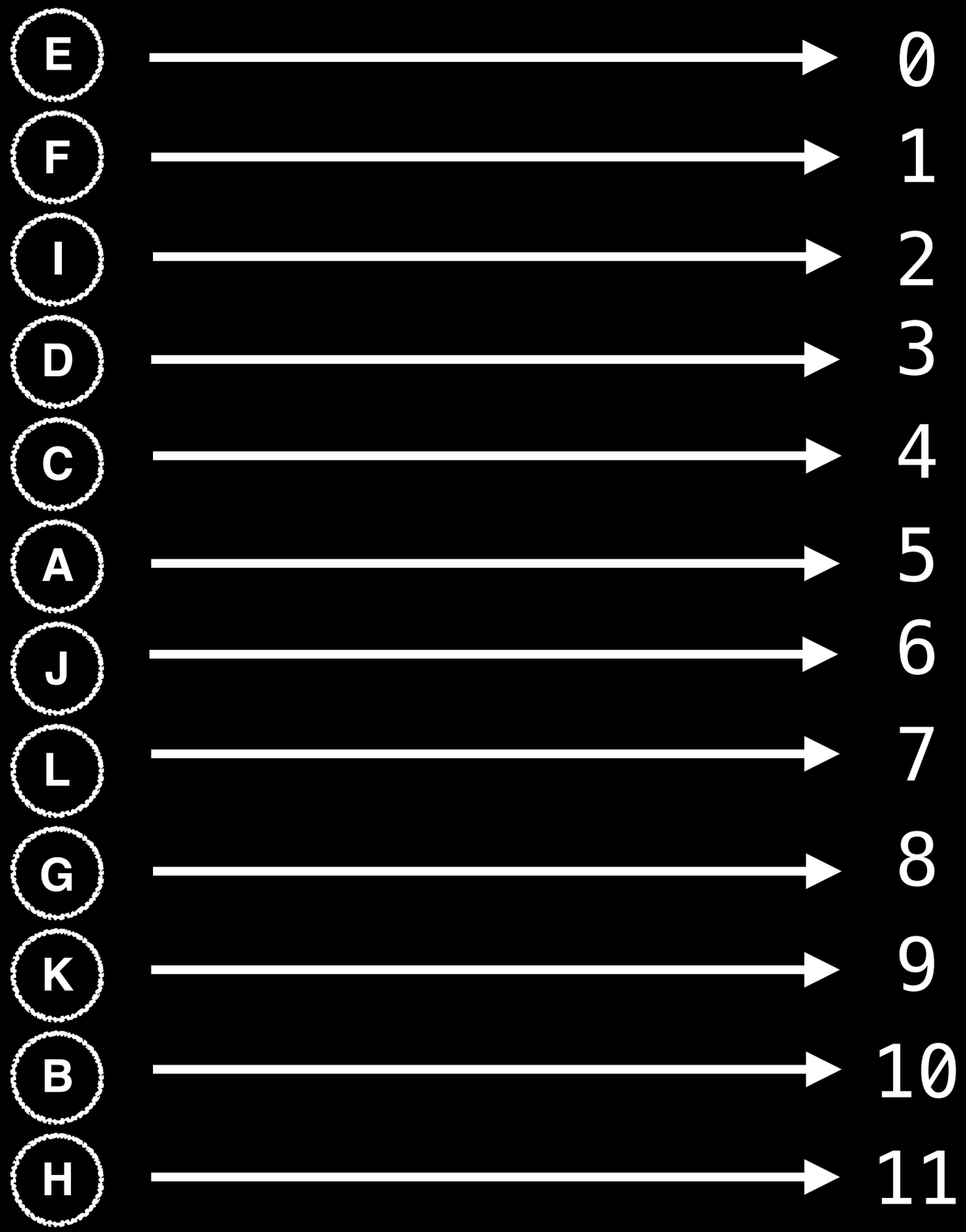
To begin using Union Find, first construct a **bijection** (a mapping) between your objects and the integers in the range [0, n).

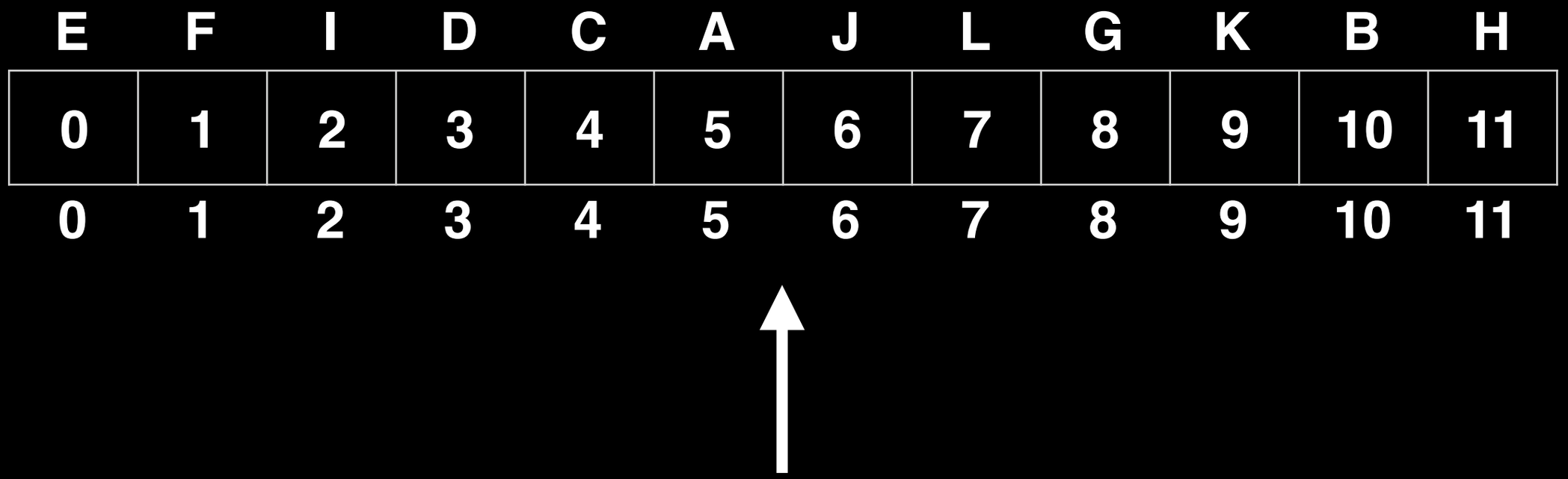
NOTE: This step is not necessary in general, but it will allow us to construct an array-based union find.

Randomly assign a mapping between the objects and the integers on the right.



0
1
2
3
4
5
6
7
8
9
10
11


E
F
I
D
C
A
J
L
G
K
B
H
0
1
2
3
4
5
6
7
8
9
10
11

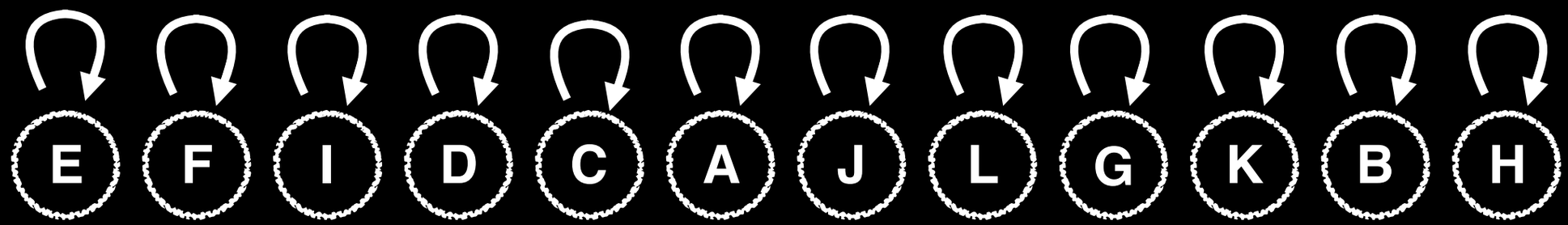


Store Union Find information in an array. Each index has an associated object (letter in this example) we can lookup through our mapping.

E	F	I	D	C	A	J	L	G	K	B	H
0	1	2	3	4	5	6	7	8	9	10	11
0	1	2	3	4	5	6	7	8	9	10	11

Instructions:

```
Union(C,K)
Union(F,E)
Union(A,J)
Union(A,B)
Union(C,D)
Union(D,I)
Union(L,F)
Union(C,A)
Union(A,B)
Union(H,G)
Union(H,F)
Union(H,B)
```

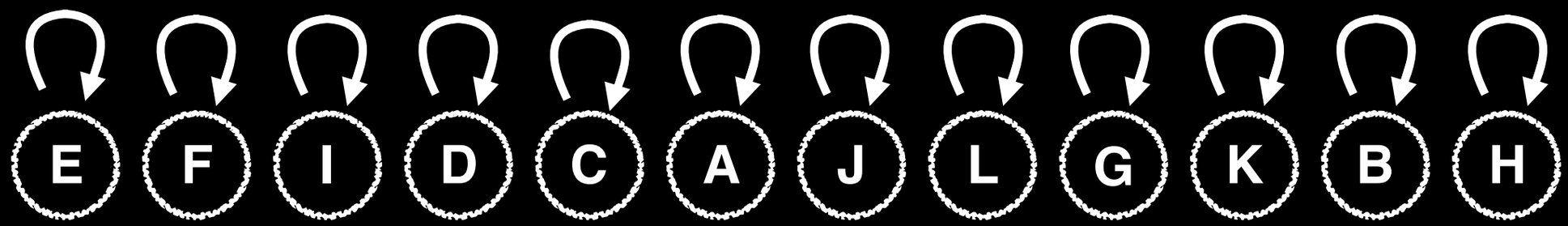


(This example does not use path compression)

E	F	I	D	C	A	J	L	G	K	B	H
0	1	2	3	4	5	6	7	8	9	10	11
0	1	2	3	4	5	6	7	8	9	10	11

Instructions:

Union(C,K)←
Union(F,E)
Union(A,J)
Union(A,B)
Union(C,D)
Union(D,I)
Union(L,F)
Union(C,A)
Union(A,B)
Union(H,G)
Union(H,F)
Union(H,B)

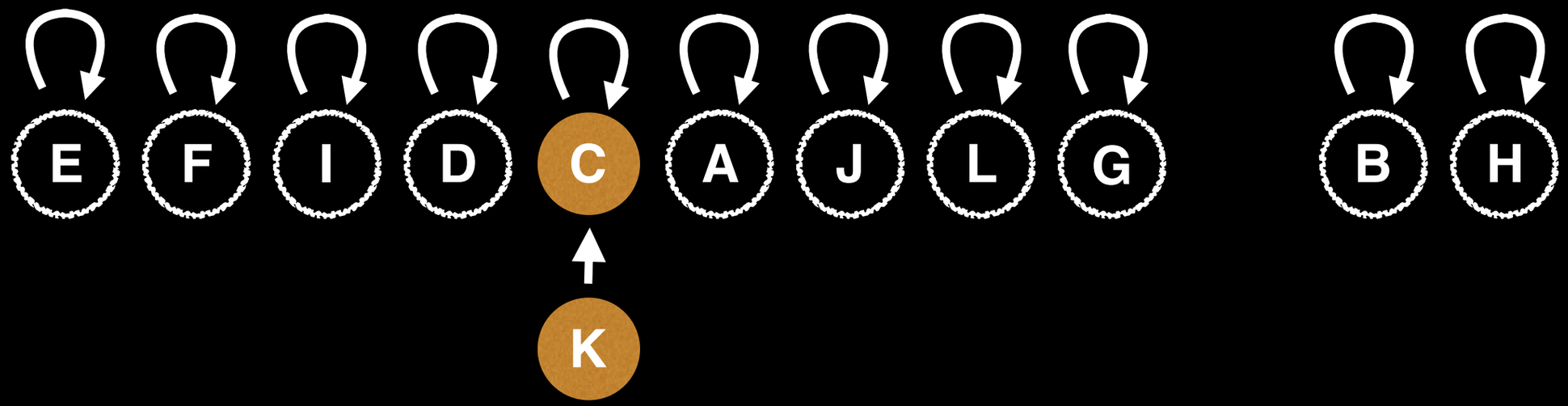


(This example does not use path compression)

E	F	I	D	C	A	J	L	G	K	B	H
0	1	2	3	4	5	6	7	8	4	10	11
0	1	2	3	4	5	6	7	8	9	10	11

Instructions:

```
Union(C,K)
Union(F,E)
Union(A,J)
Union(A,B)
Union(C,D)
Union(D,I)
Union(L,F)
Union(C,A)
Union(A,B)
Union(H,G)
Union(H,F)
Union(H,B)
```

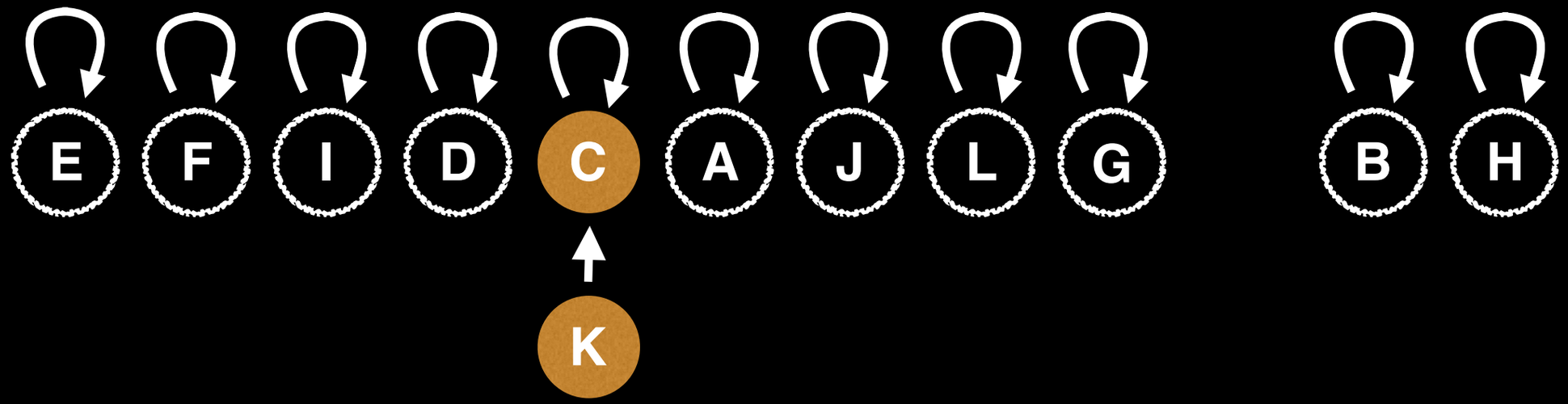


(This example does not use path compression)

E	F	I	D	C	A	J	L	G	K	B	H
0	1	2	3	4	5	6	7	8	4	10	11
0	1	2	3	4	5	6	7	8	9	10	11

Instructions:

```
Union(C,K)
Union(F,E)
Union(A,J)
Union(A,B)
Union(C,D)
Union(D,I)
Union(L,F)
Union(C,A)
Union(A,B)
Union(H,G)
Union(H,F)
Union(H,B)
```

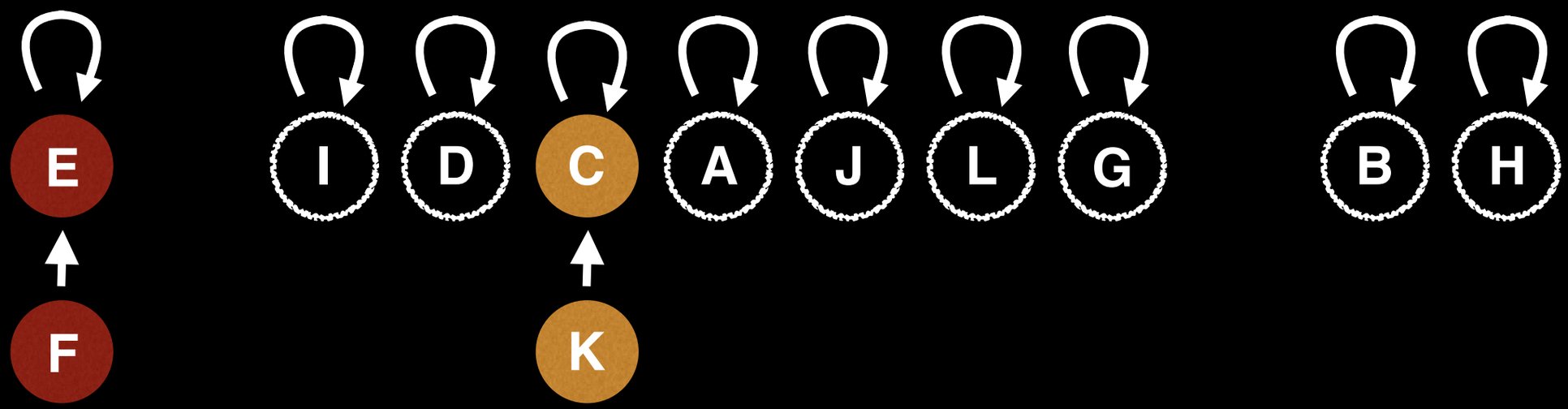


(This example does not use path compression)

E	F	I	D	C	A	J	L	G	K	B	H
0	0	2	3	4	5	6	7	8	4	10	11
0	1	2	3	4	5	6	7	8	9	10	11

Instructions:

```
Union(C,K)
Union(F,E)
Union(A,J)
Union(A,B)
Union(C,D)
Union(D,I)
Union(L,F)
Union(C,A)
Union(A,B)
Union(H,G)
Union(H,F)
Union(H,B)
```

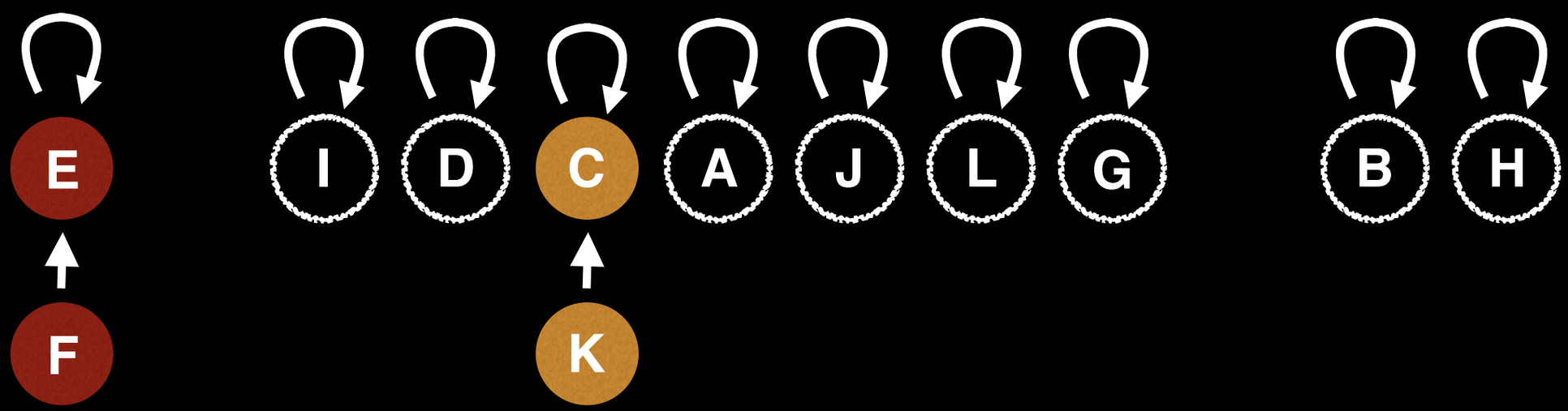


(This example does not use path compression)

E	F	I	D	C	A	J	L	G	K	B	H
0	0	2	3	4	5	6	7	8	4	10	11
0	1	2	3	4	5	6	7	8	9	10	11

Instructions:

```
Union(C,K)
Union(F,E)
Union(A,J)
Union(A,B)
Union(C,D)
Union(D,I)
Union(L,F)
Union(C,A)
Union(A,B)
Union(H,G)
Union(H,F)
Union(H,B)
```

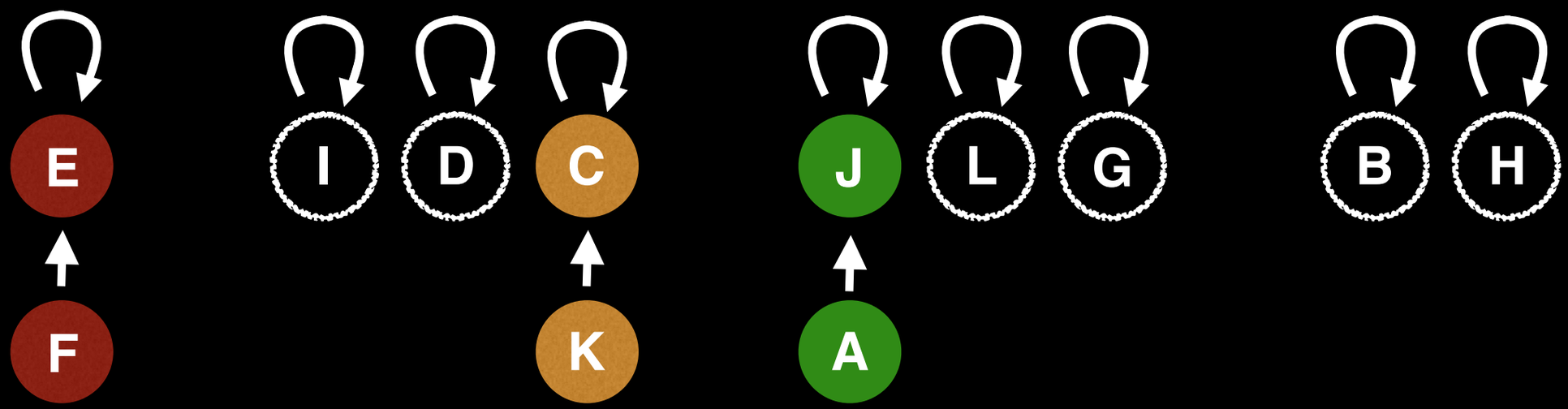


(This example does not use path compression)

E	F	I	D	C	A	J	L	G	K	B	H
0	0	2	3	4	6	6	7	8	4	10	11
0	1	2	3	4	5	6	7	8	9	10	11

Instructions:

```
Union(C,K)
Union(F,E)
Union(A,J)
Union(A,B)
Union(C,D)
Union(D,I)
Union(L,F)
Union(C,A)
Union(A,B)
Union(H,G)
Union(H,F)
Union(H,B)
```

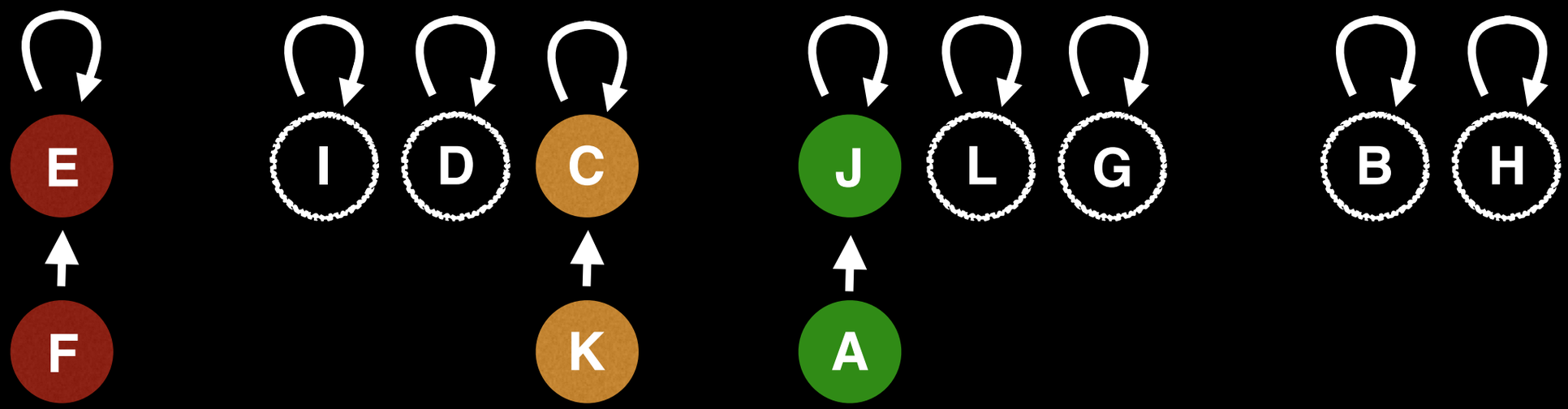


(This example does not use path compression)

E	F	I	D	C	A	J	L	G	K	B	H
0	0	2	3	4	6	6	7	8	4	10	11
0	1	2	3	4	5	6	7	8	9	10	11

Instructions:

```
Union(C,K)
Union(F,E)
Union(A,J)
Union(A,B)
Union(C,D)
Union(D,I)
Union(L,F)
Union(C,A)
Union(A,B)
Union(H,G)
Union(H,F)
Union(H,B)
```

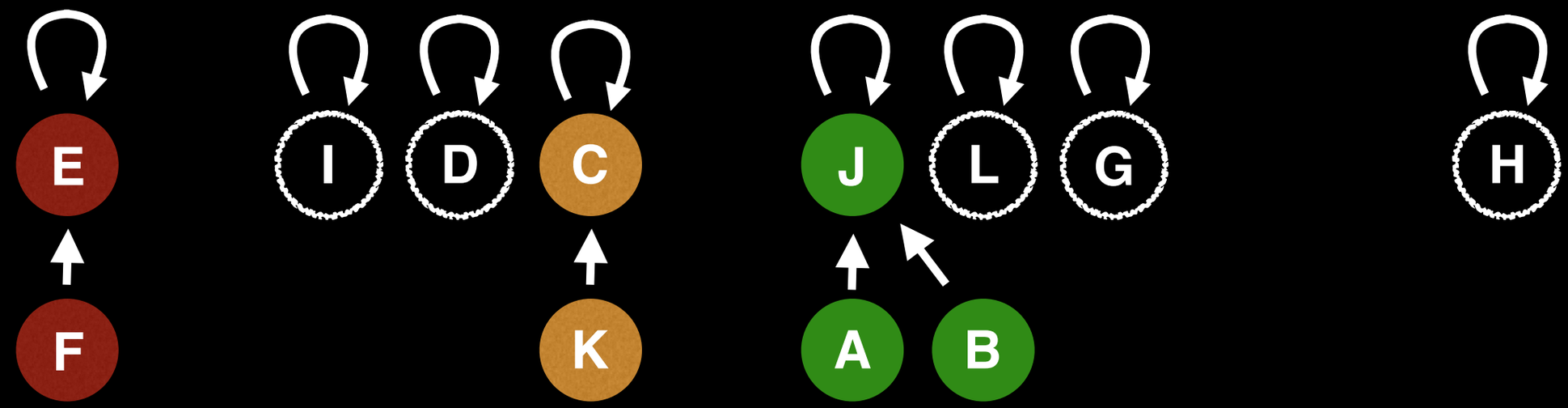


(This example does not use path compression)

E	F	I	D	C	A	J	L	G	K	B	H
0	0	2	3	4	6	6	7	8	4	6	11
0	1	2	3	4	5	6	7	8	9	10	11

Instructions:

```
Union(C,K)
Union(F,E)
Union(A,J)
Union(A,B)
Union(C,D)
Union(D,I)
Union(L,F)
Union(C,A)
Union(A,B)
Union(H,G)
Union(H,F)
Union(H,B)
```

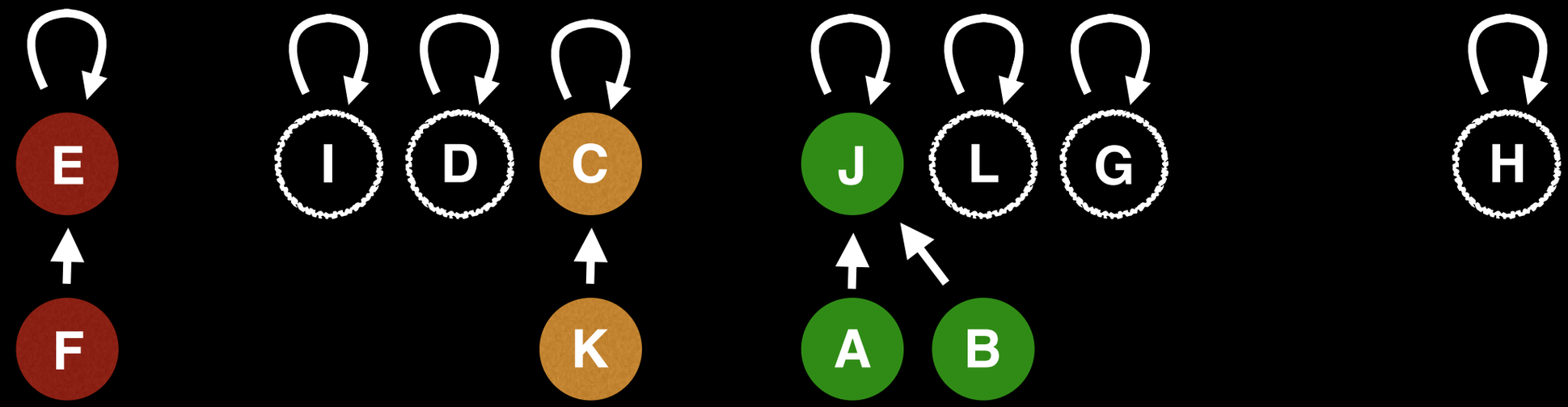


(This example does not use path compression)

E	F	I	D	C	A	J	L	G	K	B	H
0	0	2	3	4	6	6	7	8	4	6	11
0	1	2	3	4	5	6	7	8	9	10	11

Instructions:

```
Union(C,K)
Union(F,E)
Union(A,J)
Union(A,B)
Union(C,D)
Union(D,I)
Union(L,F)
Union(C,A)
Union(A,B)
Union(H,G)
Union(H,F)
Union(H,B)
```

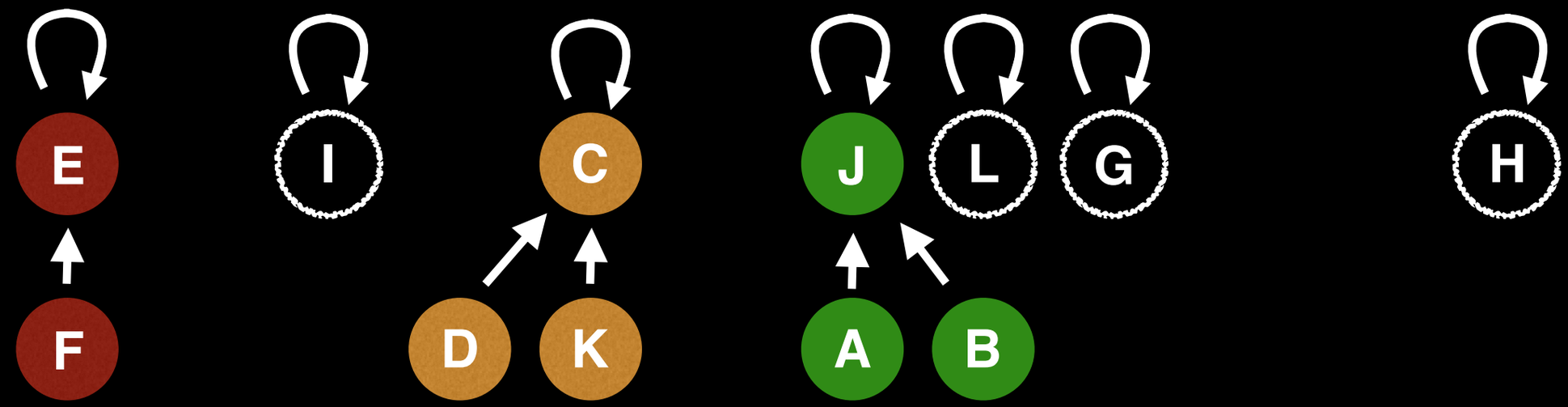


(This example does not use path compression)

E	F	I	D	C	A	J	L	G	K	B	H
0	0	2	4	4	6	6	7	8	4	6	11
0	1	2	3	4	5	6	7	8	9	10	11

Instructions:

```
Union(C,K)
Union(F,E)
Union(A,J)
Union(A,B)
Union(C,D)
Union(D,I)
Union(L,F)
Union(C,A)
Union(A,B)
Union(H,G)
Union(H,F)
Union(H,B)
```

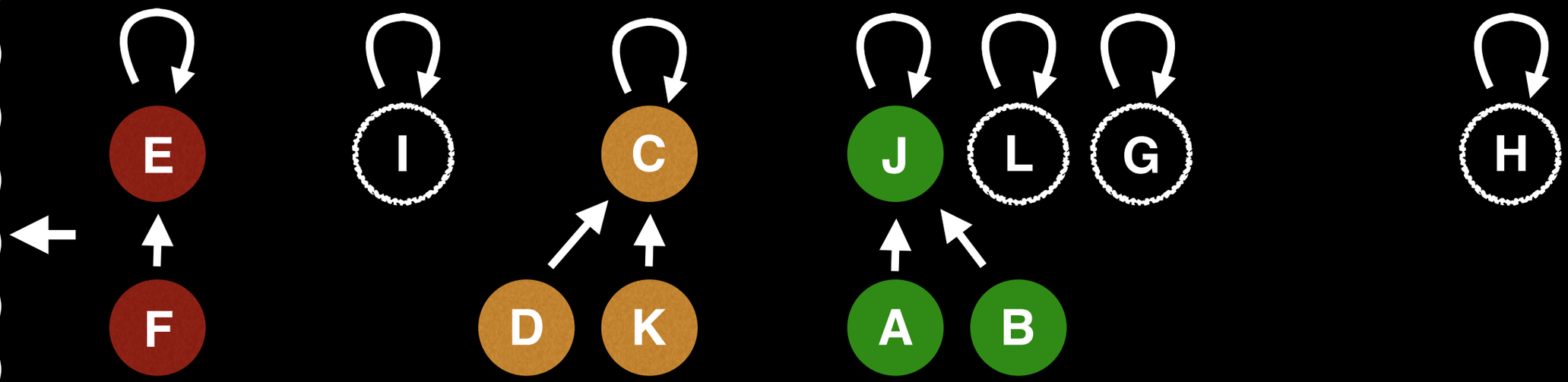


(This example does not use path compression)

E	F	I	D	C	A	J	L	G	K	B	H
0	0	2	4	4	6	6	7	8	4	6	11
0	1	2	3	4	5	6	7	8	9	10	11

Instructions:

```
Union(C,K)
Union(F,E)
Union(A,J)
Union(A,B)
Union(C,D)
Union(D,I)
Union(L,F)
Union(C,A)
Union(A,B)
Union(H,G)
Union(H,F)
Union(H,B)
```

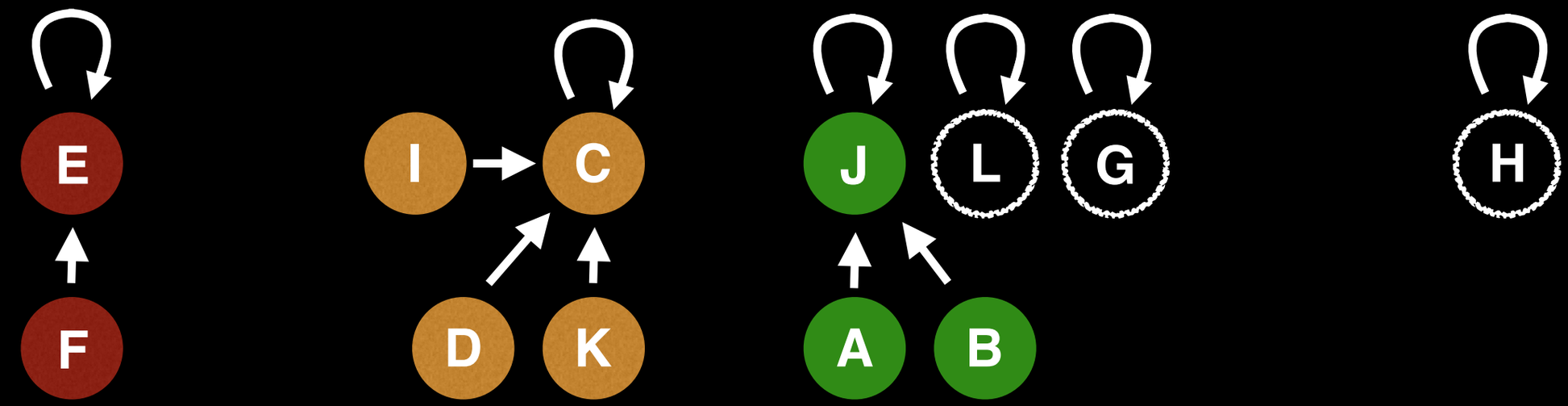


(This example does not use path compression)

E	F	I	D	C	A	J	L	G	K	B	H
0	0	4	4	4	6	6	7	8	4	6	11
0	1	2	3	4	5	6	7	8	9	10	11

Instructions:

Union(C,K)
Union(F,E)
Union(A,J)
Union(A,B)
Union(C,D)
Union(D,I)
Union(L,F)
Union(C,A)
Union(A,B)
Union(H,G)
Union(H,F)
Union(H,B)

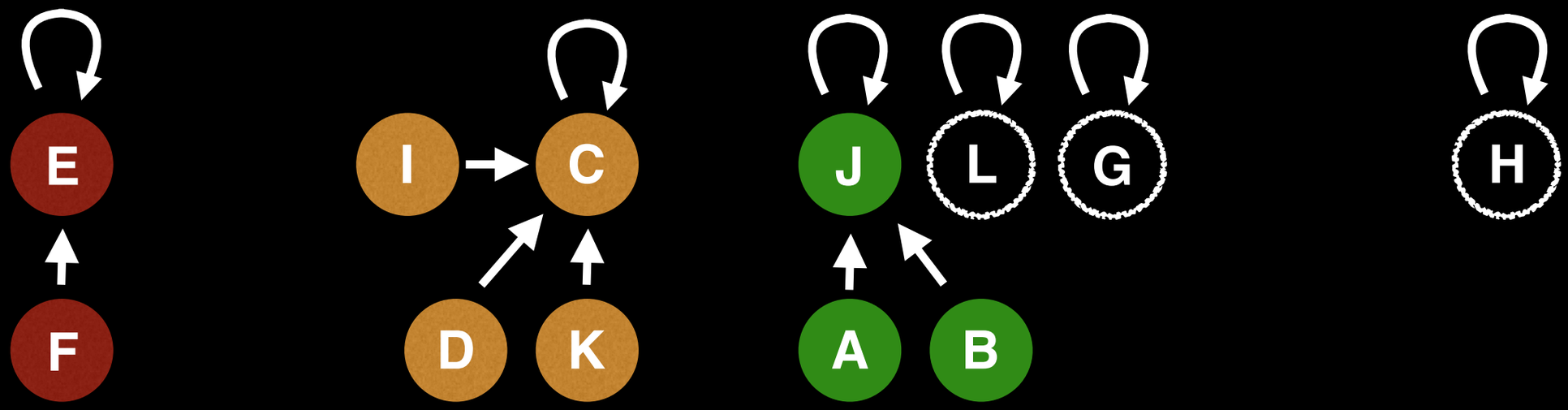


(This example does not use path compression)

E	F	I	D	C	A	J	L	G	K	B	H
0	0	4	4	4	6	6	7	8	4	6	11
0	1	2	3	4	5	6	7	8	9	10	11

Instructions:

```
Union(C,K)
Union(F,E)
Union(A,J)
Union(A,B)
Union(C,D)
Union(D,I)
Union(L,F)
Union(C,A)
Union(A,B)
Union(H,G)
Union(H,F)
Union(H,B)
```

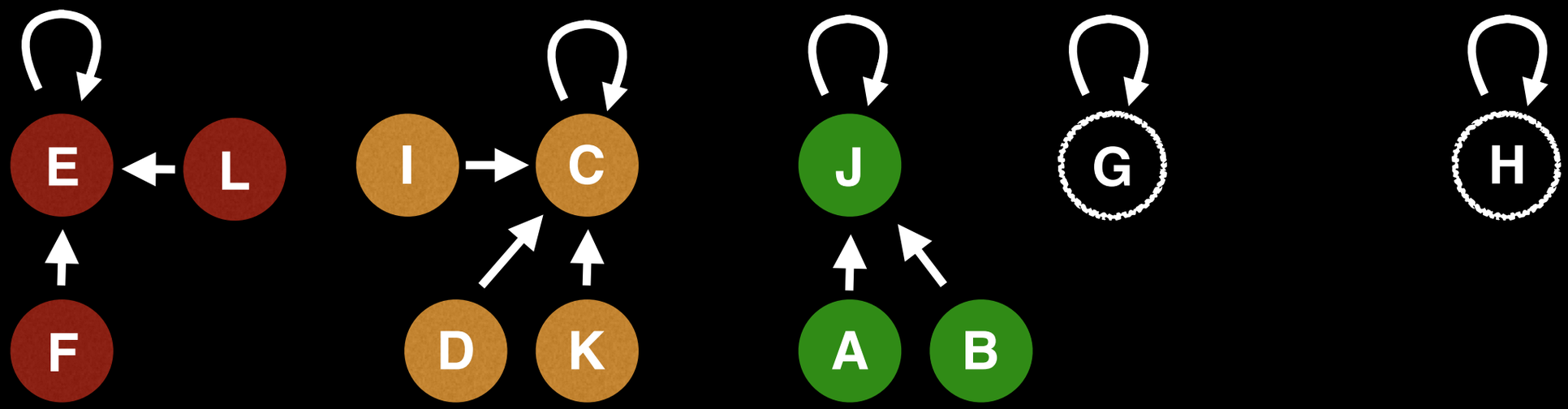


(This example does not use path compression)

E	F	I	D	C	A	J	L	G	K	B	H
0	0	4	4	4	6	6	0	8	4	6	11
0	1	2	3	4	5	6	7	8	9	10	11

Instructions:

```
Union(C,K)
Union(F,E)
Union(A,J)
Union(A,B)
Union(C,D)
Union(D,I)
Union(L,F)
Union(C,A)
Union(A,B)
Union(H,G)
Union(H,F)
Union(H,B)
```

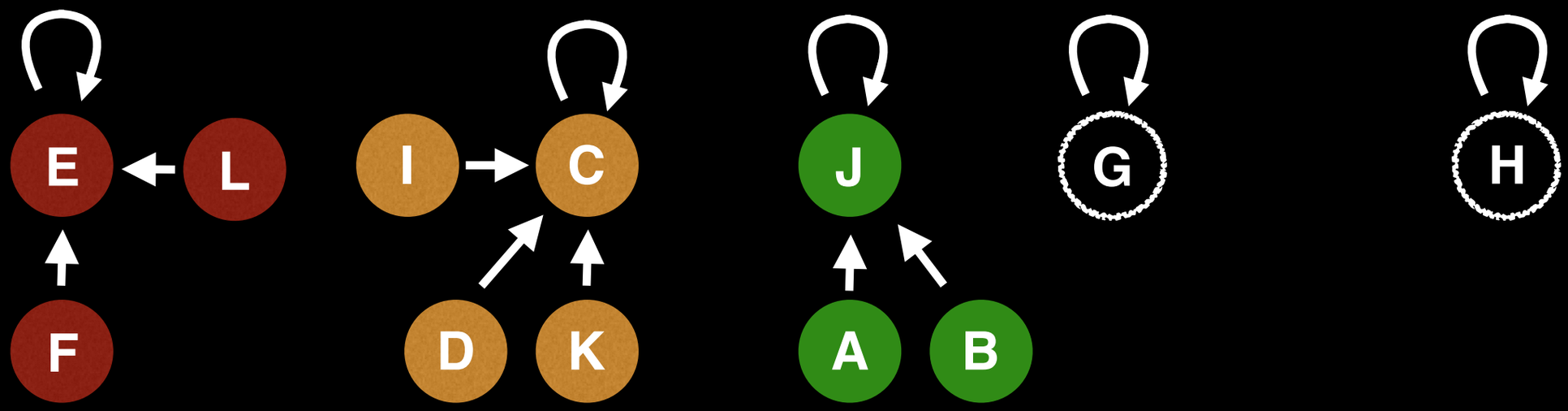


(This example does not use path compression)

E	F	I	D	C	A	J	L	G	K	B	H
0	0	4	4	4	6	6	0	8	4	6	11
0	1	2	3	4	5	6	7	8	9	10	11

Instructions:

```
Union(C,K)
Union(F,E)
Union(A,J)
Union(A,B)
Union(C,D)
Union(D,I)
Union(L,F)
Union(C,A)
Union(A,B)
Union(H,G)
Union(H,F)
Union(H,B)
```

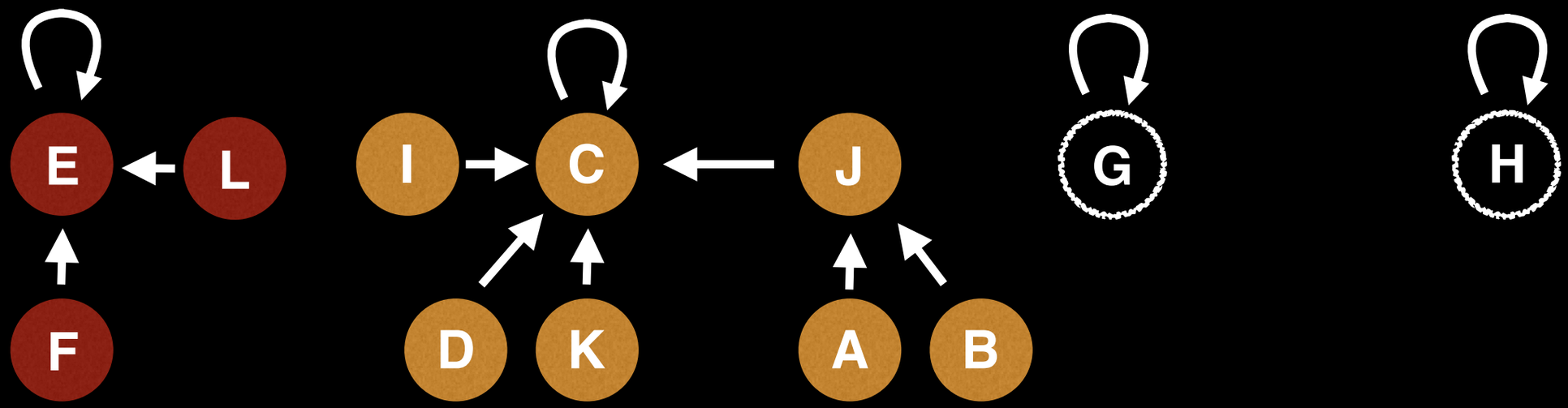


(This example does not use path compression)

E	F	I	D	C	A	J	L	G	K	B	H
0	0	4	4	4	6	4	0	8	4	6	11
0	1	2	3	4	5	6	7	8	9	10	11

Instructions:

```
Union(C,K)
Union(F,E)
Union(A,J)
Union(A,B)
Union(C,D)
Union(D,I)
Union(L,F)
Union(C,A)
Union(A,B)
Union(H,G)
Union(H,F)
Union(H,B)
```

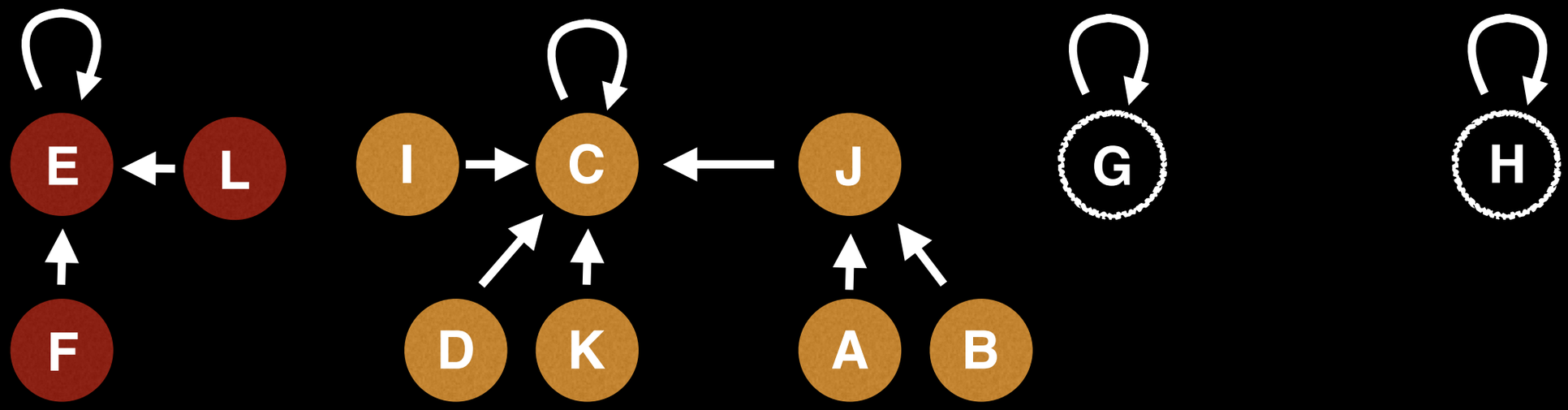


(This example does not use path compression)

E	F	I	D	C	A	J	L	G	K	B	H
0	0	4	4	4	6	4	0	8	4	6	11
0	1	2	3	4	5	6	7	8	9	10	11

Instructions:

```
Union(C,K)
Union(F,E)
Union(A,J)
Union(A,B)
Union(C,D)
Union(D,I)
Union(L,F)
Union(C,A)
Union(A,B)←
Union(H,G)
Union(H,F)
Union(H,B)
```

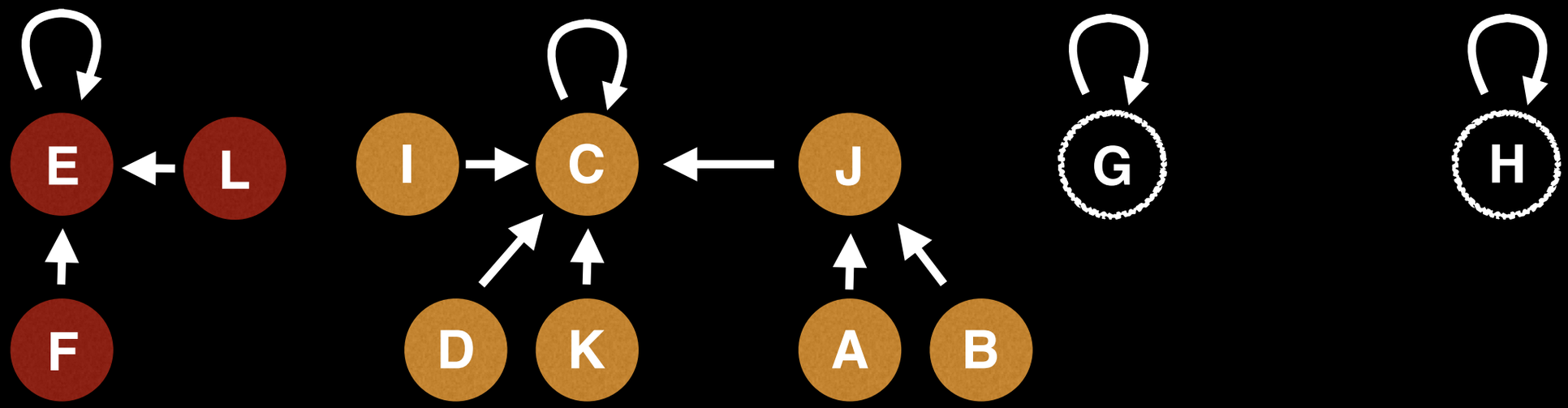


(This example does not use path compression)

E	F	I	D	C	A	J	L	G	K	B	H
0	0	4	4	4	6	4	0	8	4	6	11
0	1	2	3	4	5	6	7	8	9	10	11

Instructions:

```
Union(C,K)
Union(F,E)
Union(A,J)
Union(A,B)
Union(C,D)
Union(D,I)
Union(L,F)
Union(C,A)
Union(A,B)
Union(H,G)
Union(H,F)
Union(H,B)
```

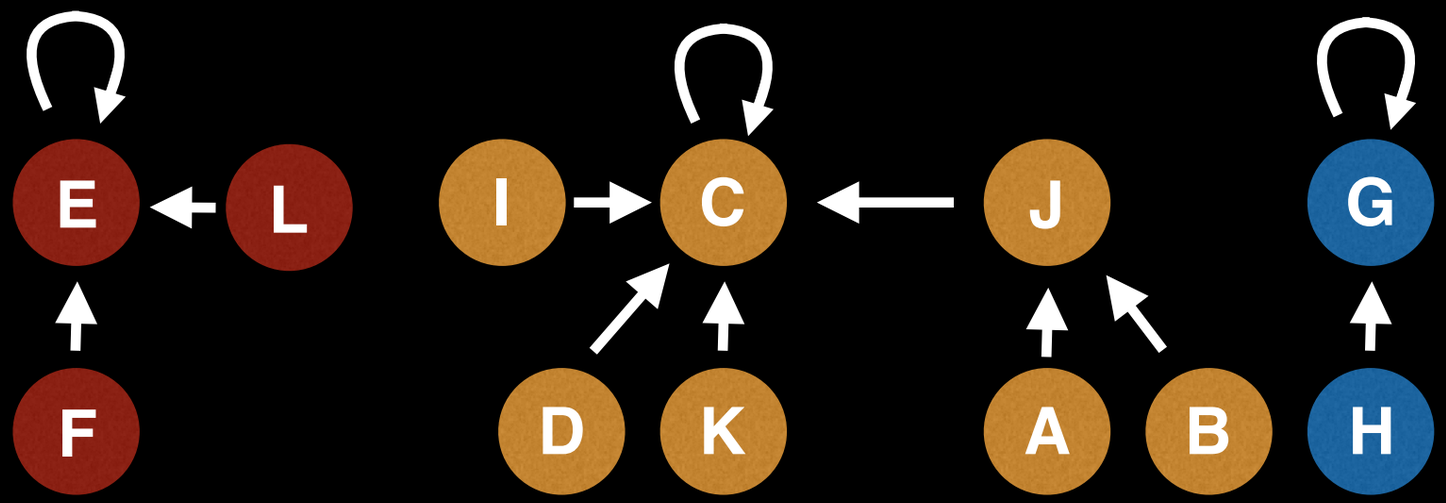


(This example does not use path compression)

E	F	I	D	C	A	J	L	G	K	B	H
0	0	4	4	4	6	4	0	8	4	6	8
0	1	2	3	4	5	6	7	8	9	10	11

Instructions:

```
Union(C,K)
Union(F,E)
Union(A,J)
Union(A,B)
Union(C,D)
Union(D,I)
Union(L,F)
Union(C,A)
Union(A,B)
Union(H,G)
Union(H,F)
Union(H,B)
```

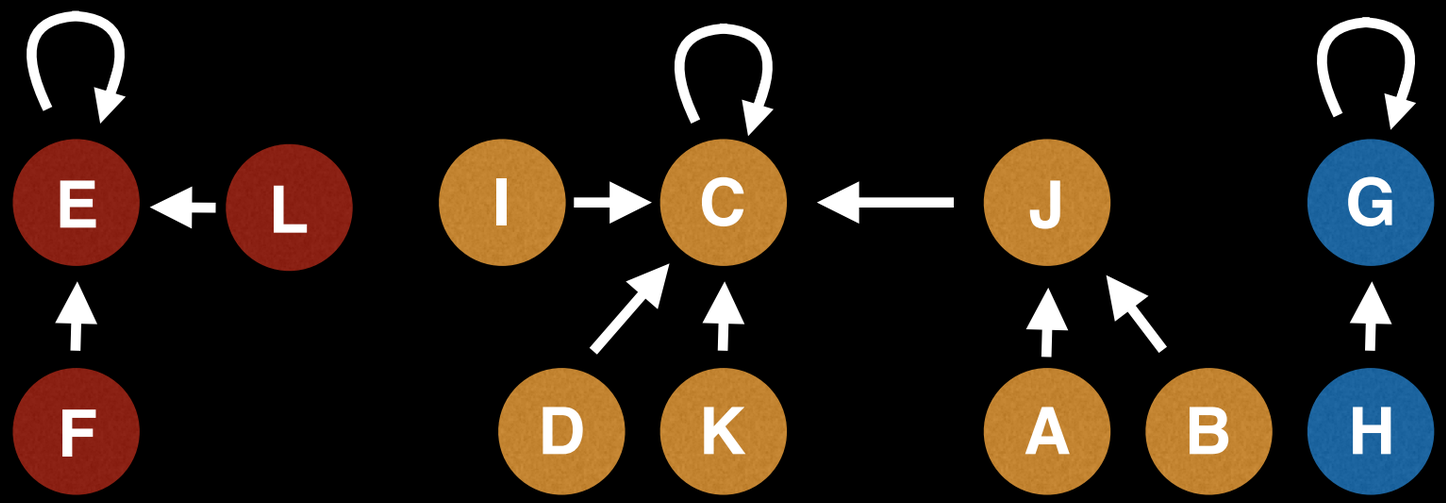


(This example does not use path compression)

E	F	I	D	C	A	J	L	G	K	B	H
0	0	4	4	4	6	4	0	8	4	6	8
0	1	2	3	4	5	6	7	8	9	10	11

Instructions:

```
Union(C,K)
Union(F,E)
Union(A,J)
Union(A,B)
Union(C,D)
Union(D,I)
Union(L,F)
Union(C,A)
Union(A,B)
Union(H,G)
Union(H,F)◀
Union(H,B)
```

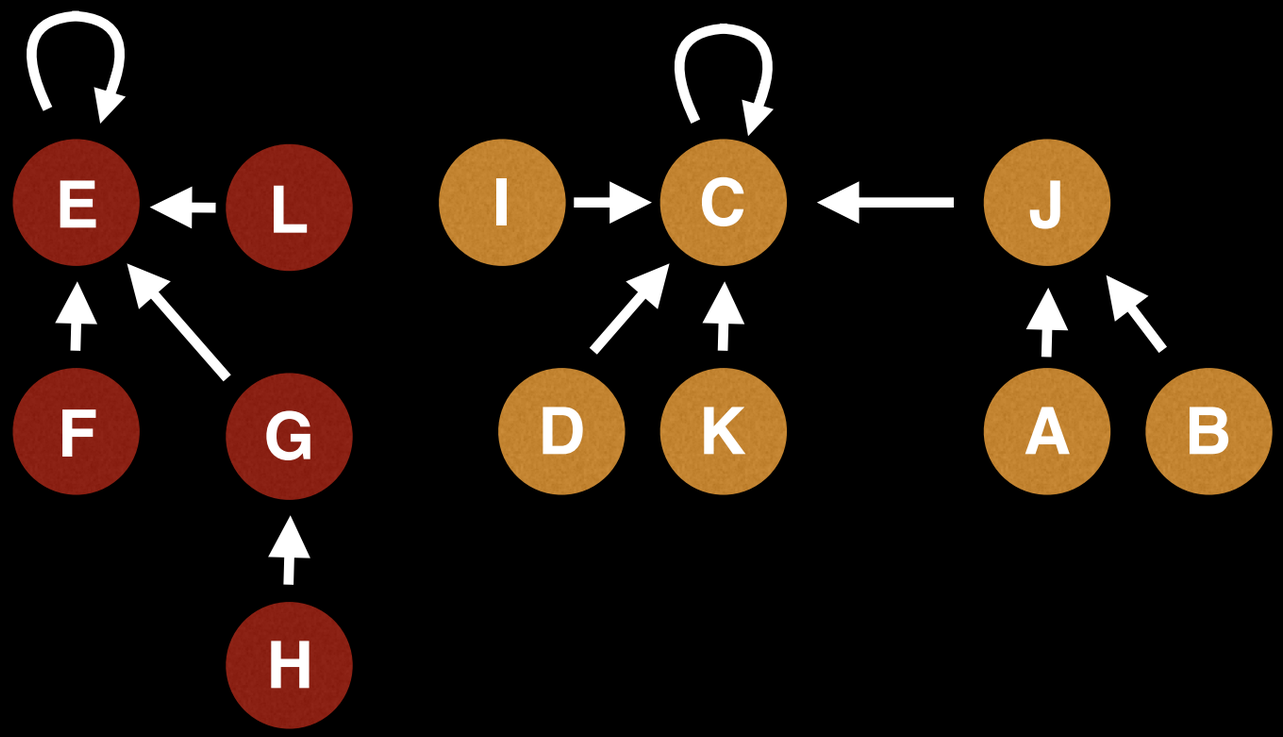


(This example does not use path compression)

E	F	I	D	C	A	J	L	G	K	B	H
0	0	4	4	4	6	4	0	0	4	6	8
0	1	2	3	4	5	6	7	8	9	10	11

Instructions:

```
Union(C,K)
Union(F,E)
Union(A,J)
Union(A,B)
Union(C,D)
Union(D,I)
Union(L,F)
Union(C,A)
Union(A,B)
Union(H,G)
Union(H,F) ←
Union(H,B)
```

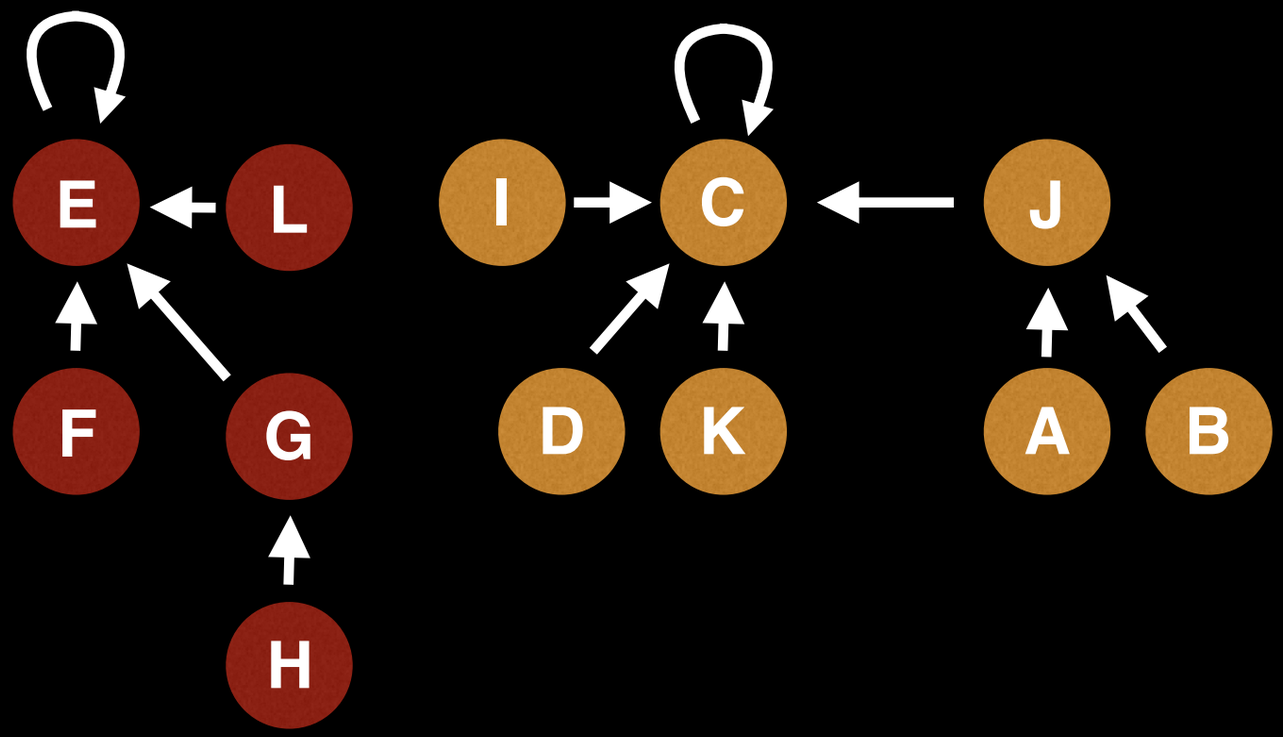


(This example does not use path compression)

E	F	I	D	C	A	J	L	G	K	B	H
0	0	4	4	4	6	4	0	0	4	6	8
0	1	2	3	4	5	6	7	8	9	10	11

Instructions:

```
Union(C,K)
Union(F,E)
Union(A,J)
Union(A,B)
Union(C,D)
Union(D,I)
Union(L,F)
Union(C,A)
Union(A,B)
Union(H,G)
Union(H,F)
Union(H,B)
```

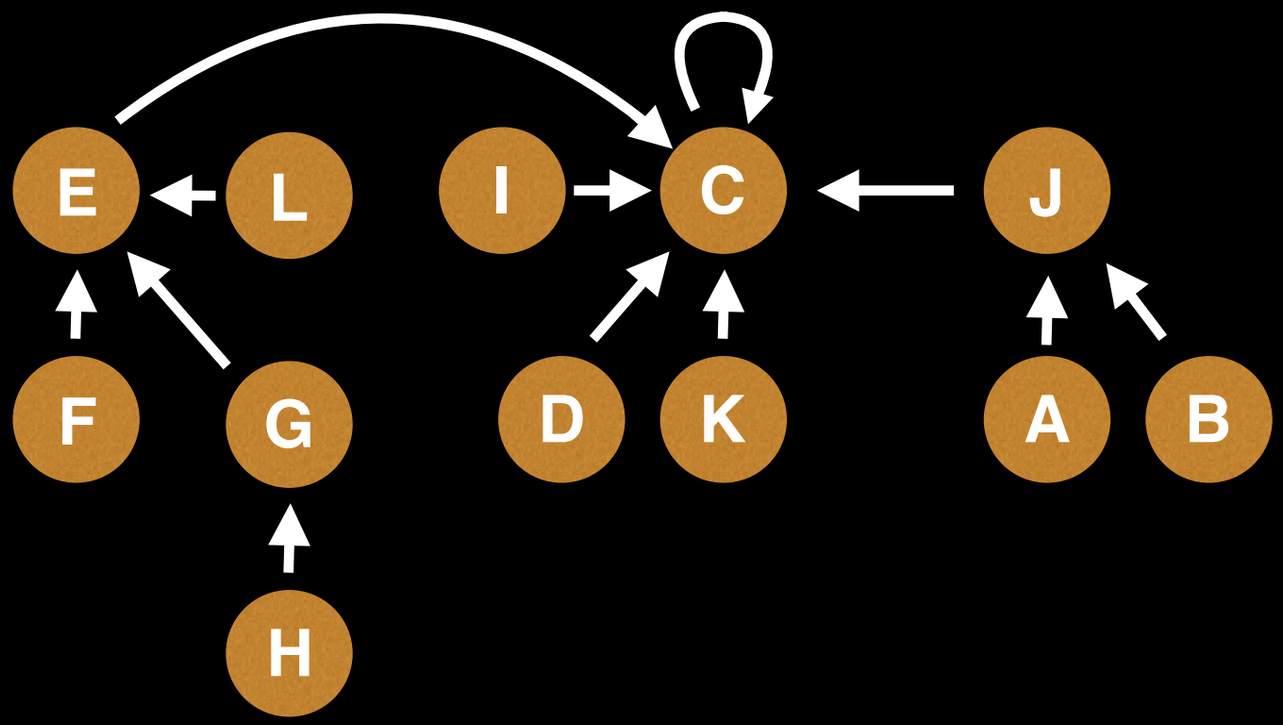


(This example does not use path compression)

E	F	I	D	C	A	J	L	G	K	B	H
4	0	4	4	4	6	4	0	0	4	6	8
0	1	2	3	4	5	6	7	8	9	10	11

Instructions:

```
Union(C,K)
Union(F,E)
Union(A,J)
Union(A,B)
Union(C,D)
Union(D,I)
Union(L,F)
Union(C,A)
Union(A,B)
Union(H,G)
Union(H,F)
Union(H,B)
```



(This example does not use path compression)

Remarks

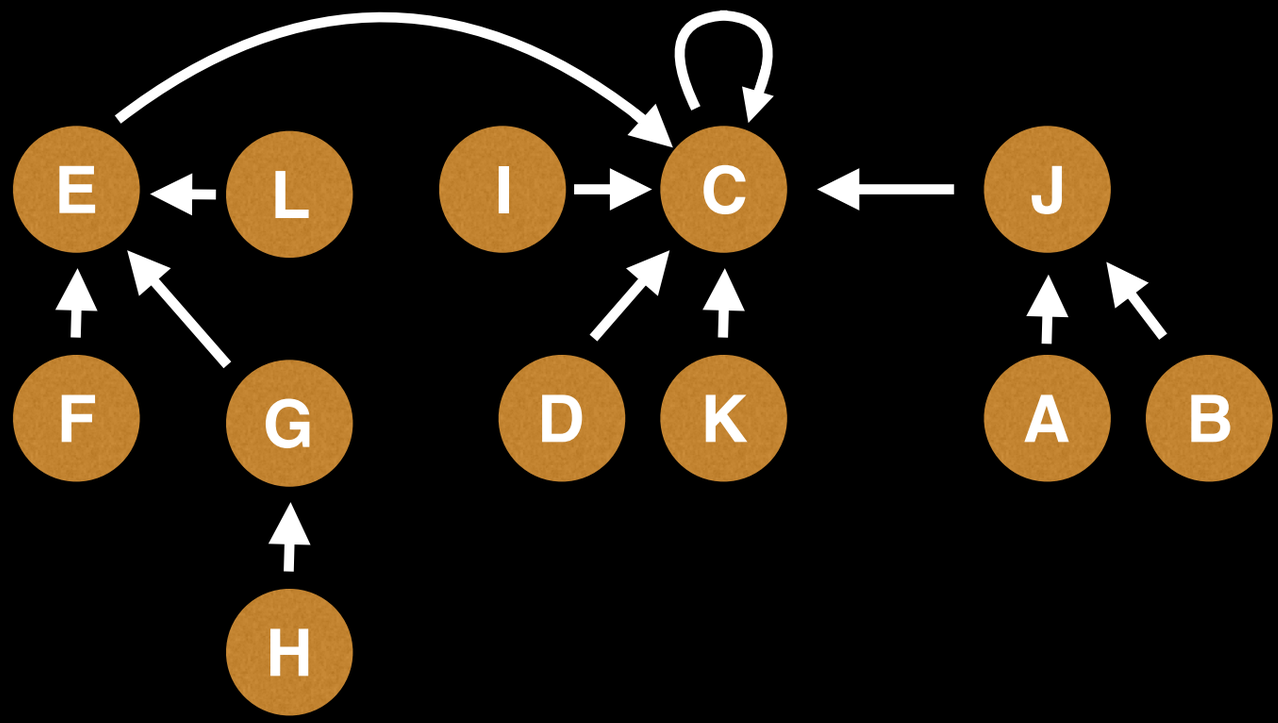
In this data structure, we do not 'un-union' elements. In general, this would be very inefficient to do since each of its possible many children of a node would need to be updated.

The number of components is equal to the number of roots remaining. Also note that the number of roots never increases.

Remarks

Our current version of Union Find does not support the nice $\alpha(1)$ time complexity we want.

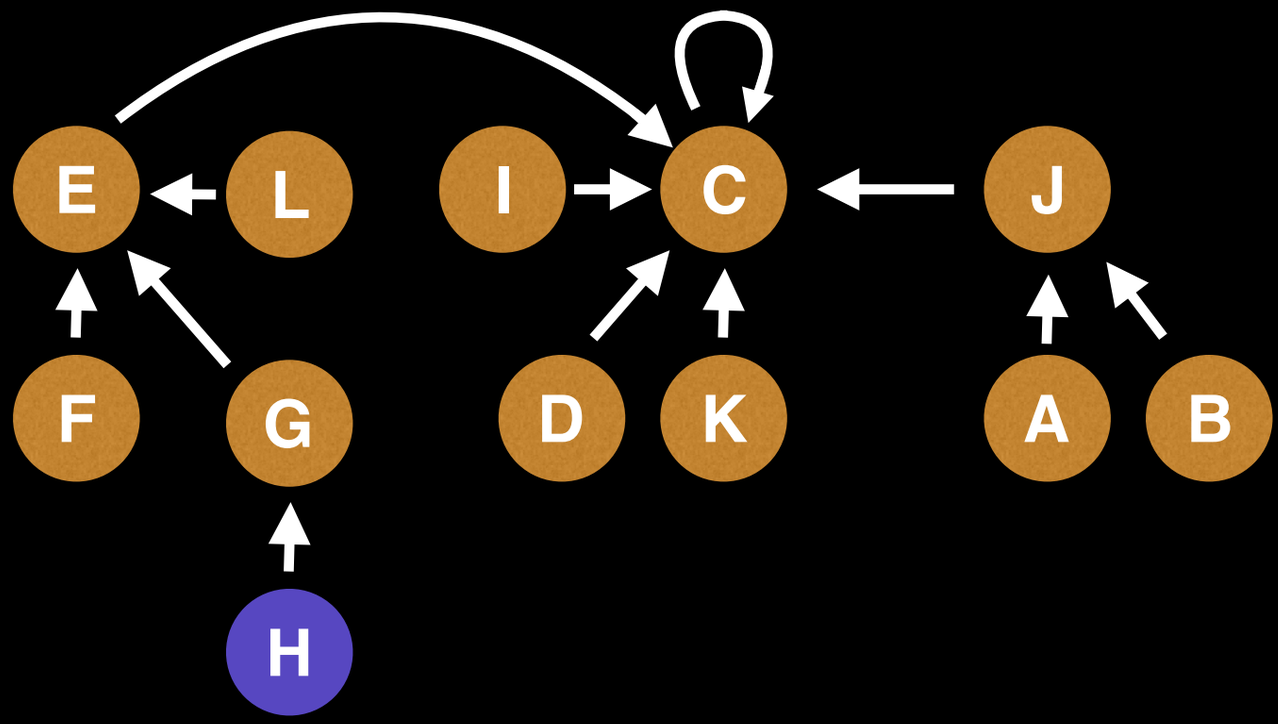
Checking if H and B belong to the same group takes five hops and in the worst case this is potentially much more.



Remarks

Our current version of Union Find does not support the nice $\alpha(1)$ time complexity we want.

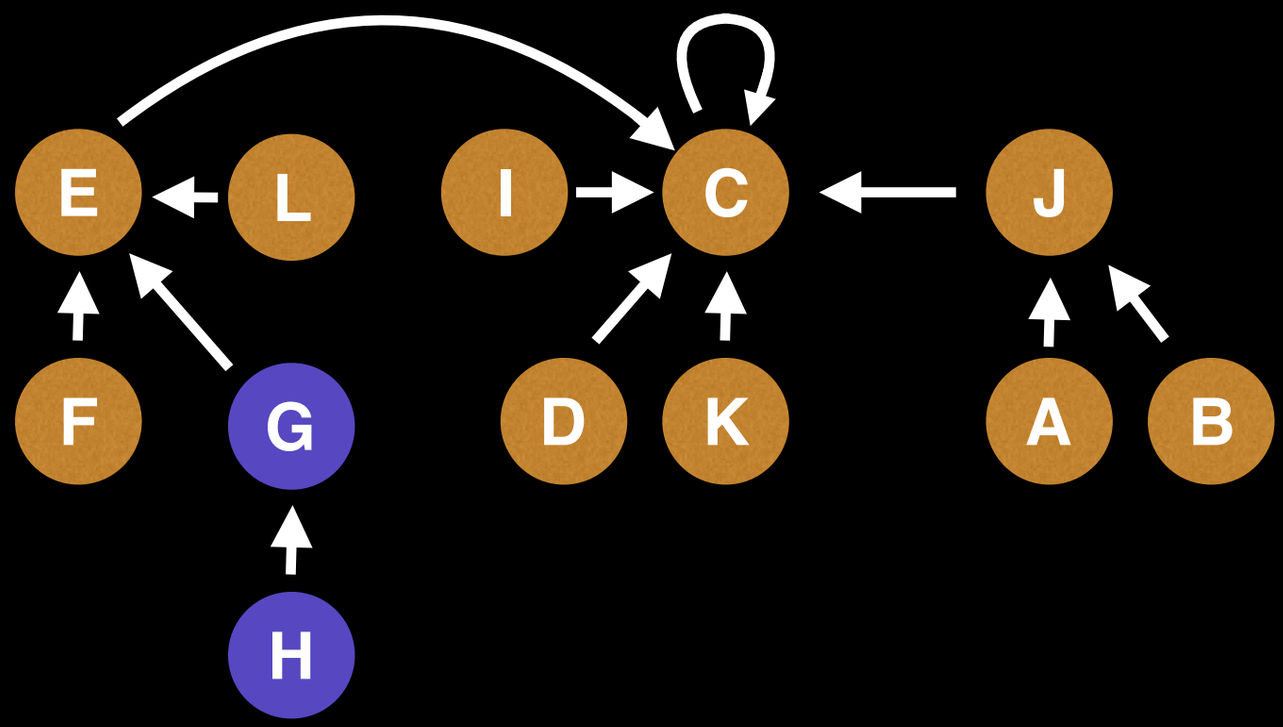
Checking if H and B belong to the same group takes five hops and in the worst case this is potentially much more.



Remarks

Our current version of Union Find does not support the nice $\alpha(1)$ time complexity we want.

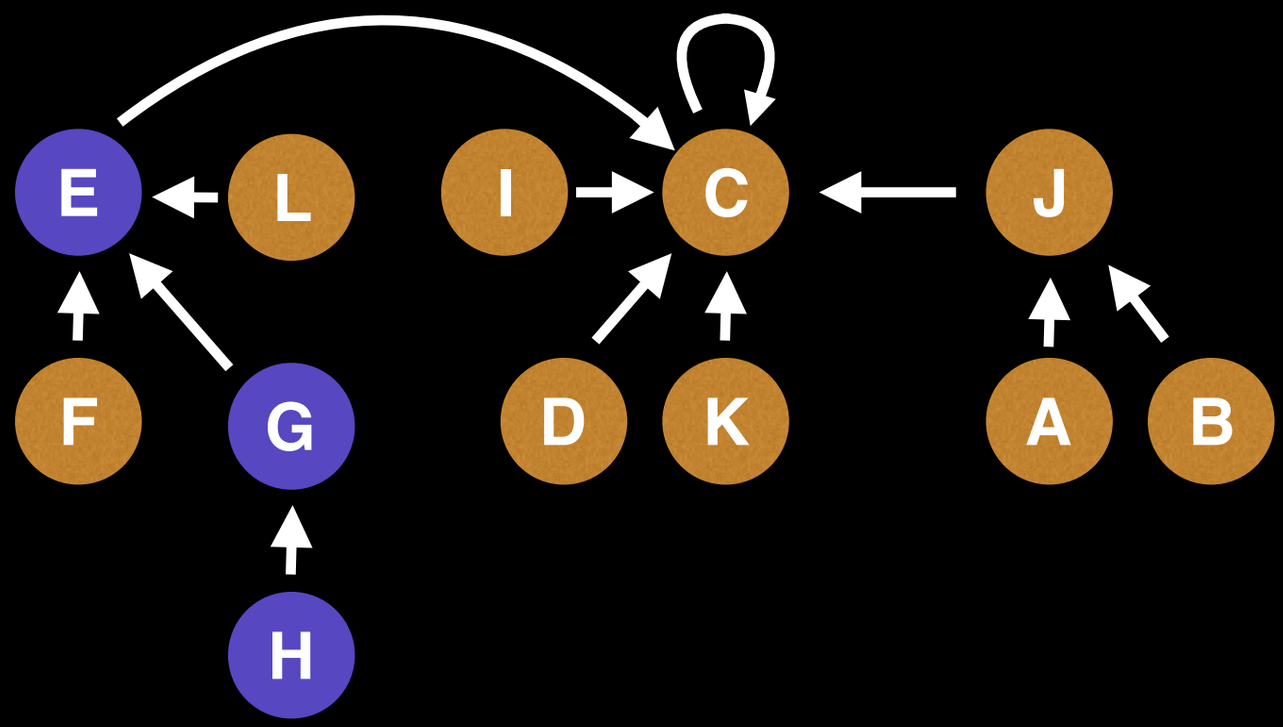
Checking if H and B belong to the same group takes five hops and in the worst case this is potentially much more.



Remarks

Our current version of Union Find does not support the nice $\alpha(1)$ time complexity we want.

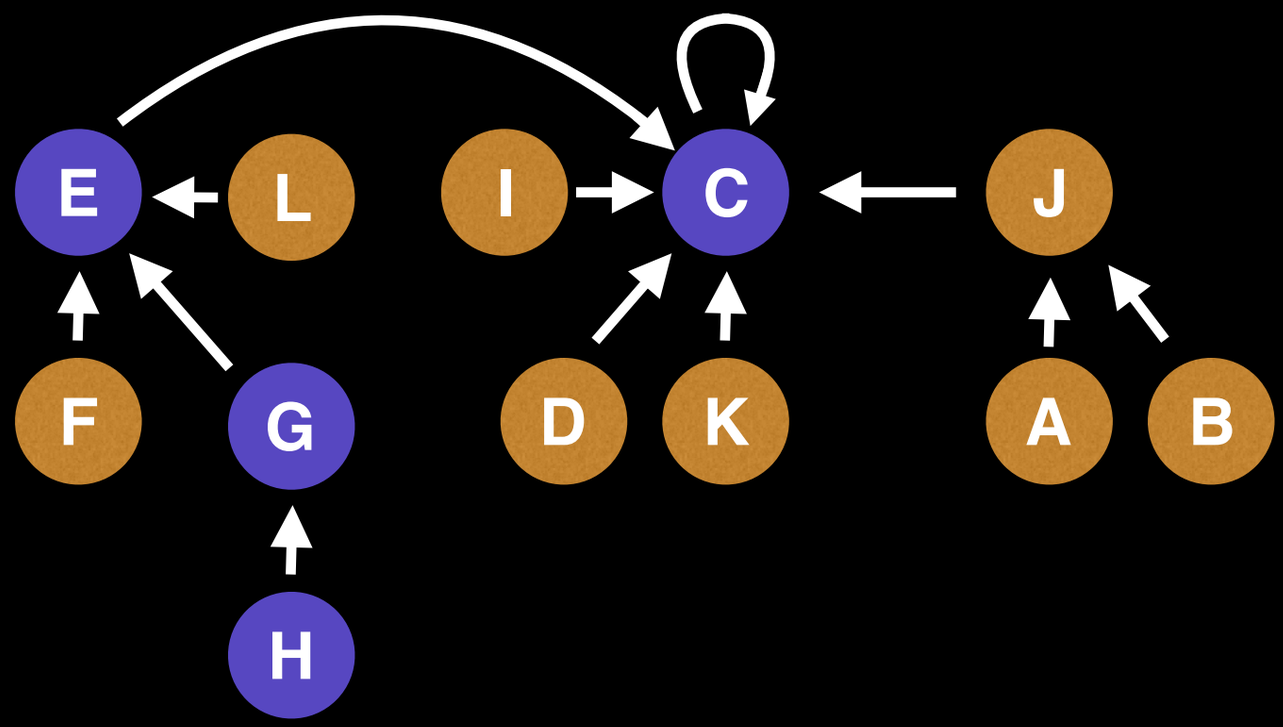
Checking if H and B belong to the same group takes five hops and in the worst case this is potentially much more.



Remarks

Our current version of Union Find does not support the nice $\alpha(1)$ time complexity we want.

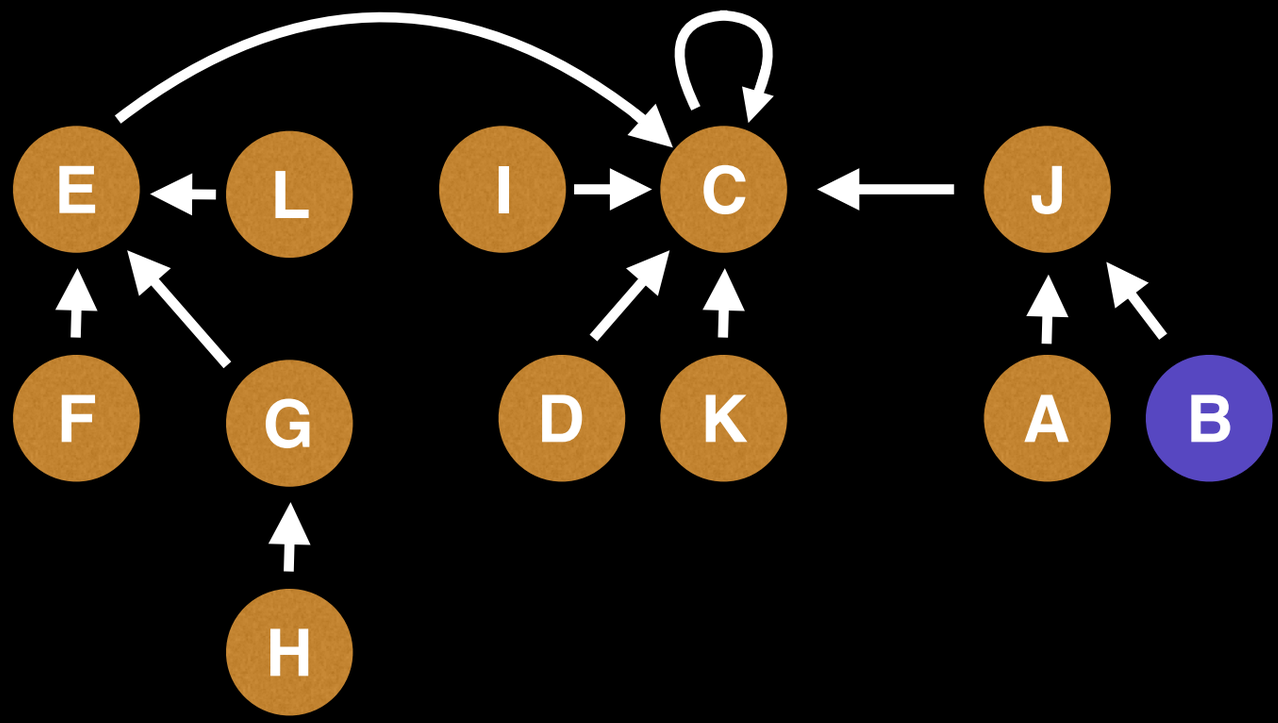
Checking if H and B belong to the same group takes five hops and in the worst case this is potentially much more.



Remarks

Our current version of Union Find does not support the nice $\alpha(1)$ time complexity we want.

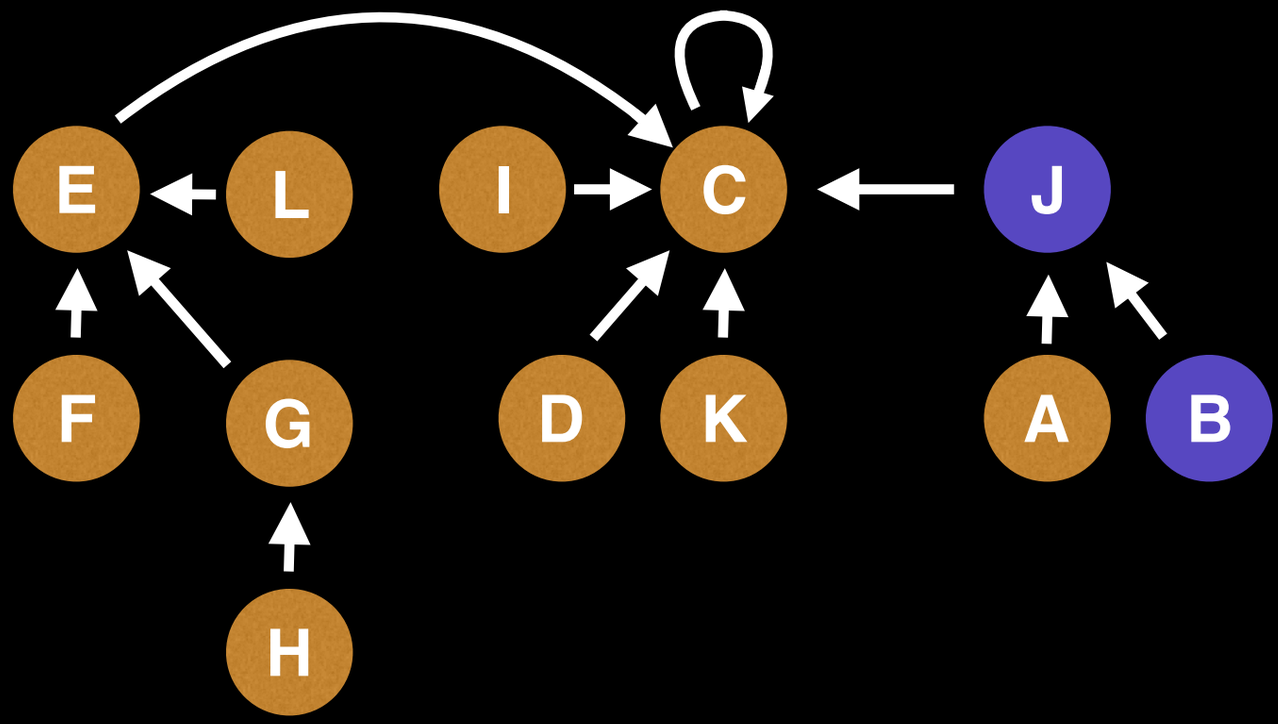
Checking if H and B belong to the same group takes five hops and in the worst case this is potentially much more.



Remarks

Our current version of Union Find does not support the nice $\alpha(1)$ time complexity we want.

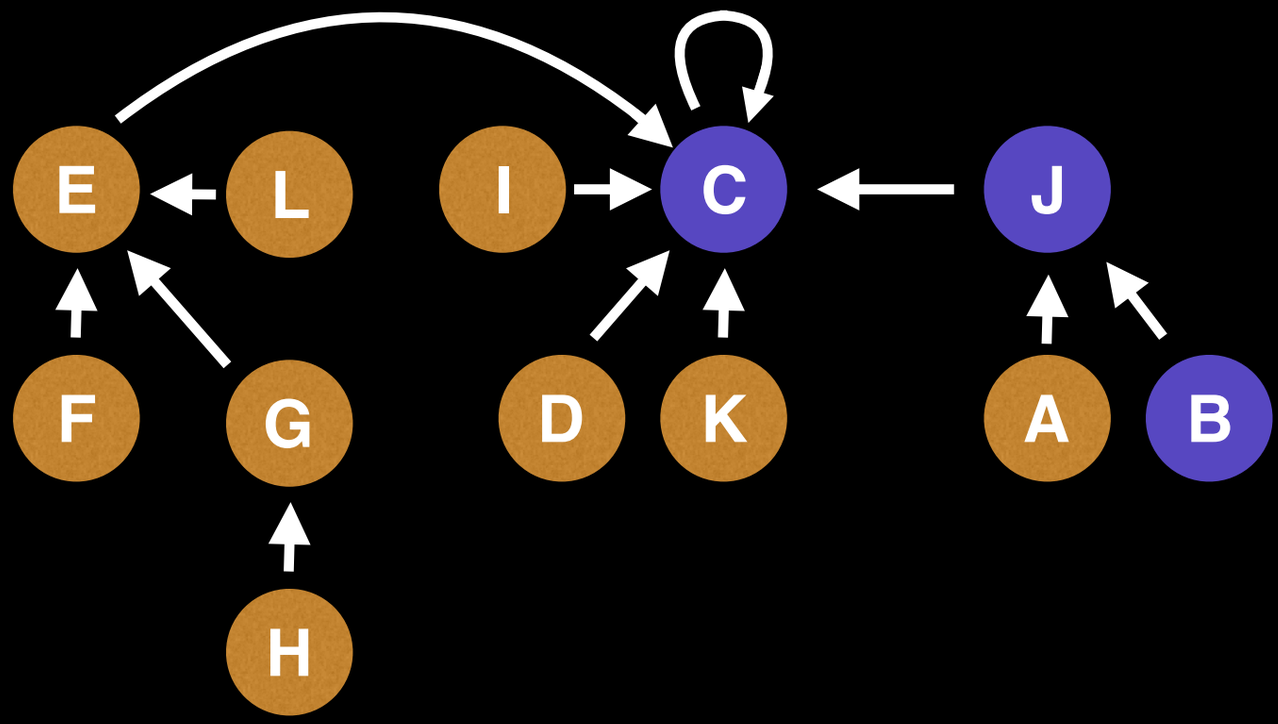
Checking if H and B belong to the same group takes five hops and in the worst case this is potentially much more.



Remarks

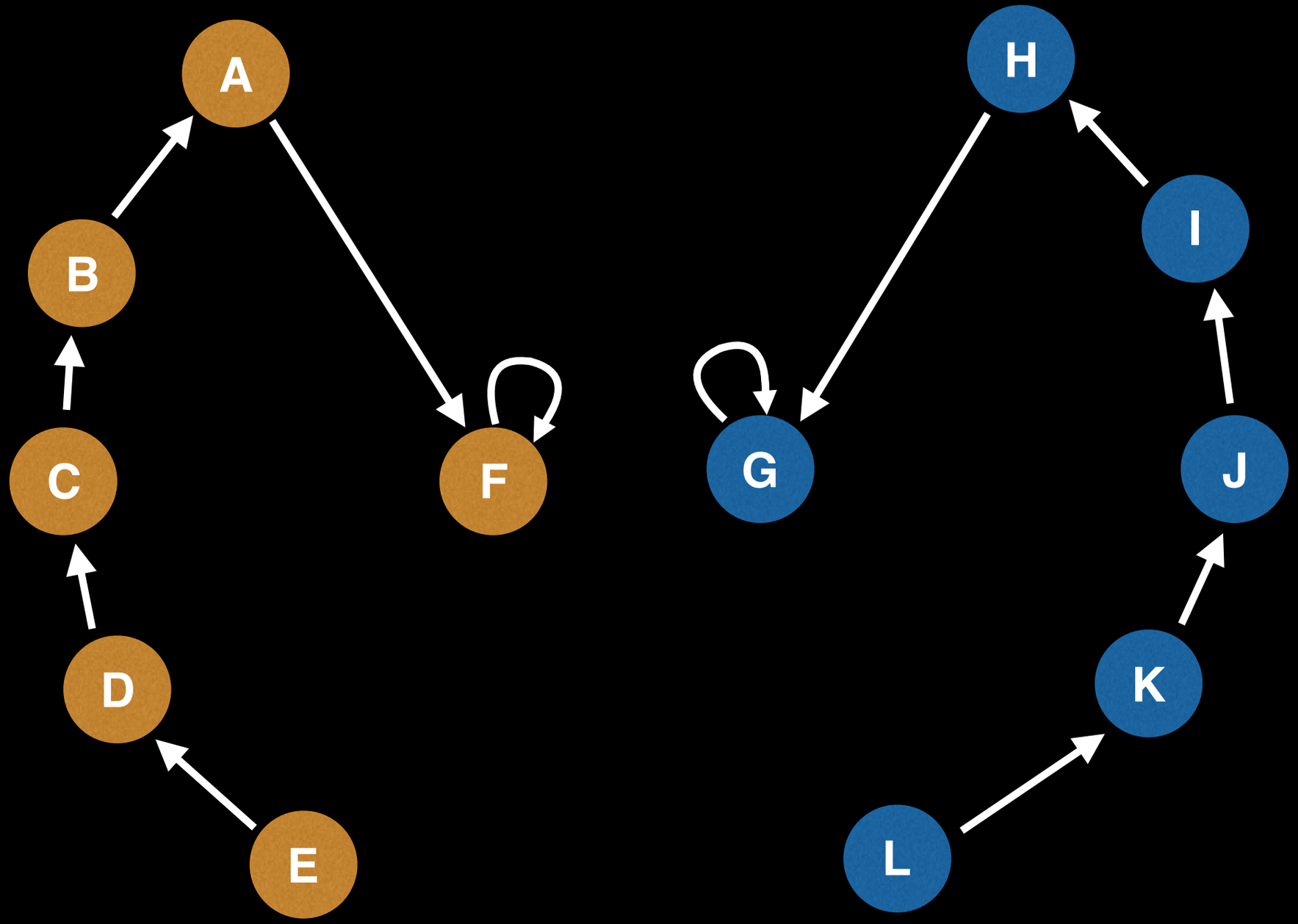
Our current version of Union Find does not support the nice $\alpha(1)$ time complexity we want.

Checking if H and B belong to the same group takes five hops and in the worst case this is potentially much more.



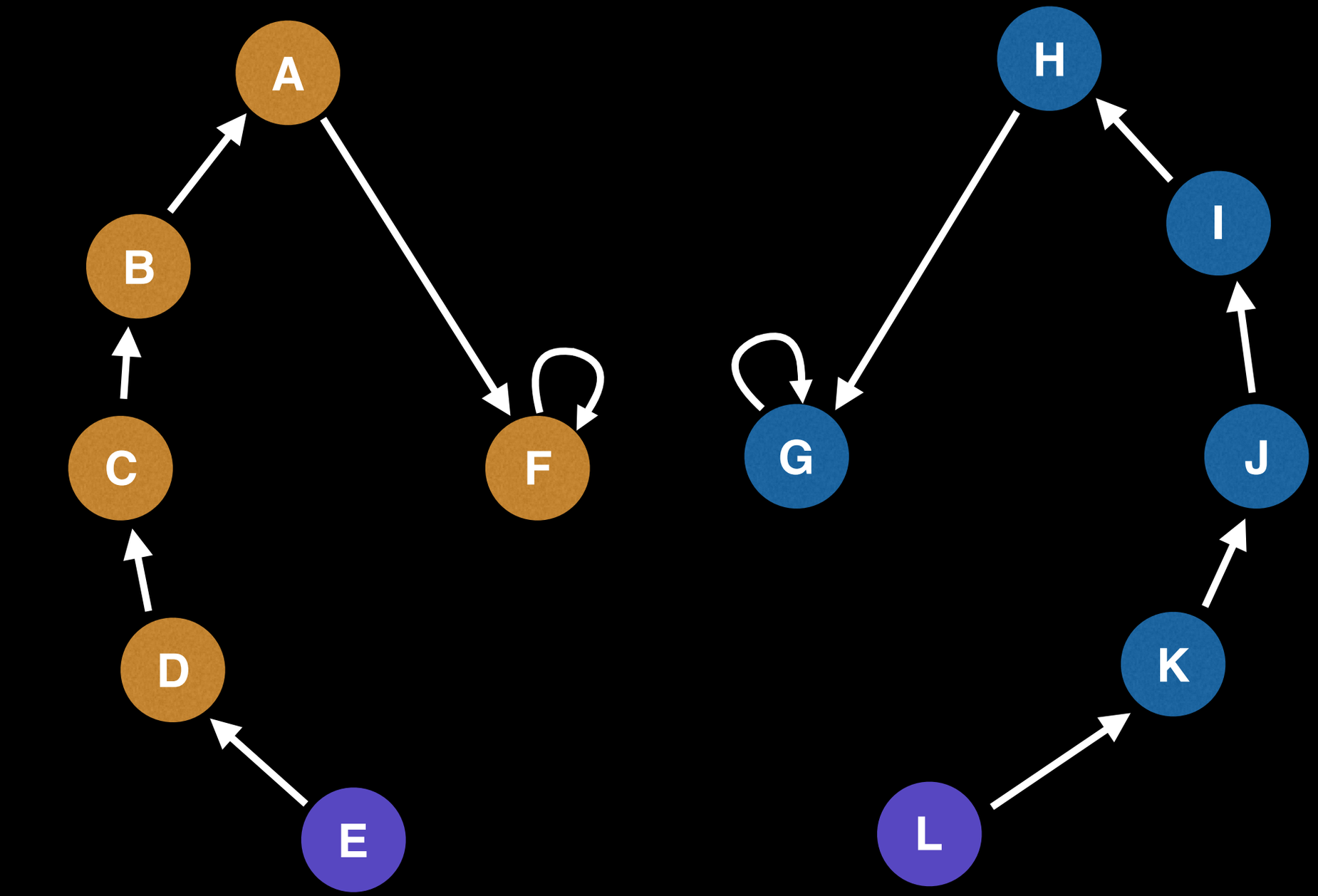
Path Compression Union Find

Hypothetical Union Find path compression example



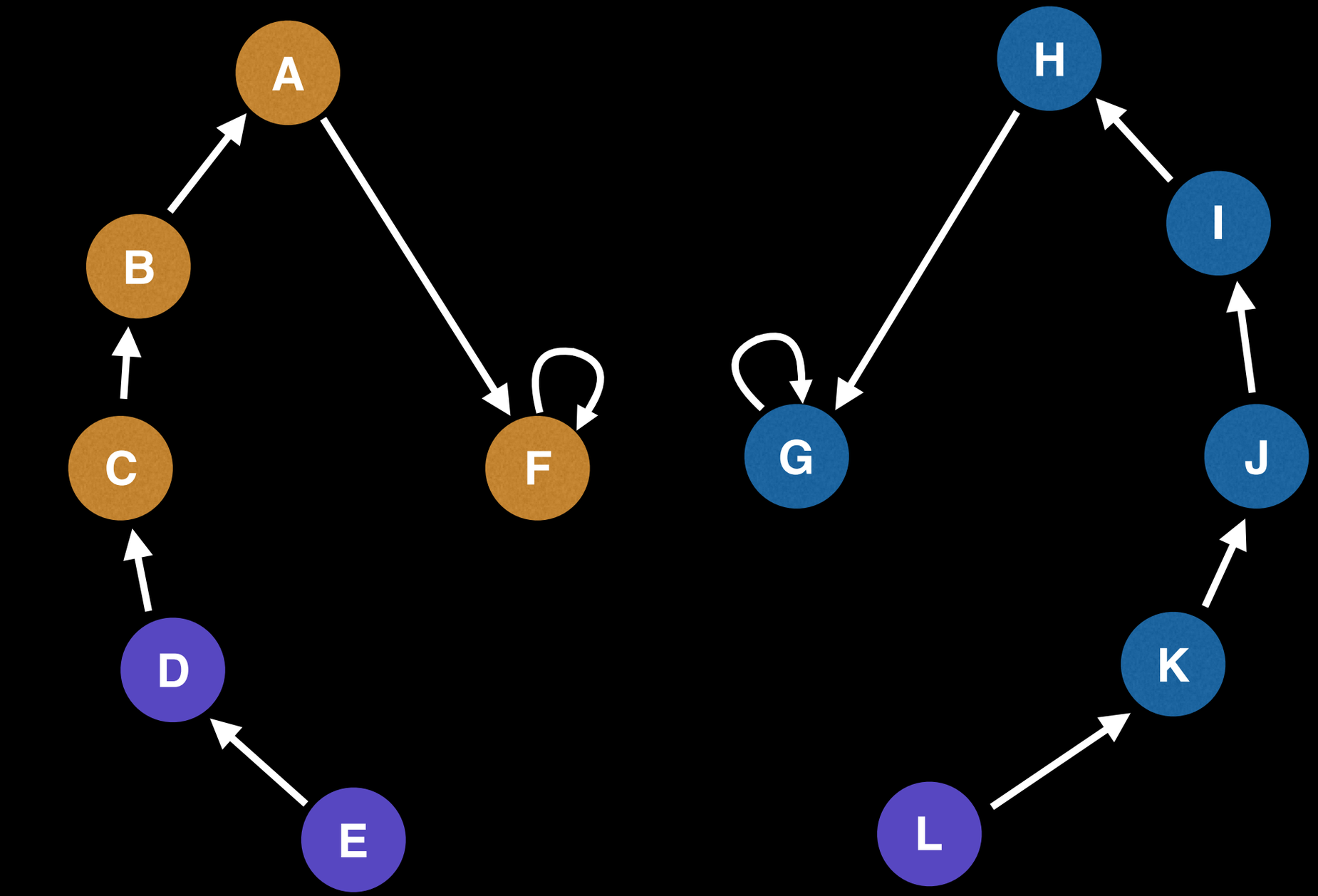
Operation: Take the union of **E** and **L**

Hypothetical Union Find path compression example



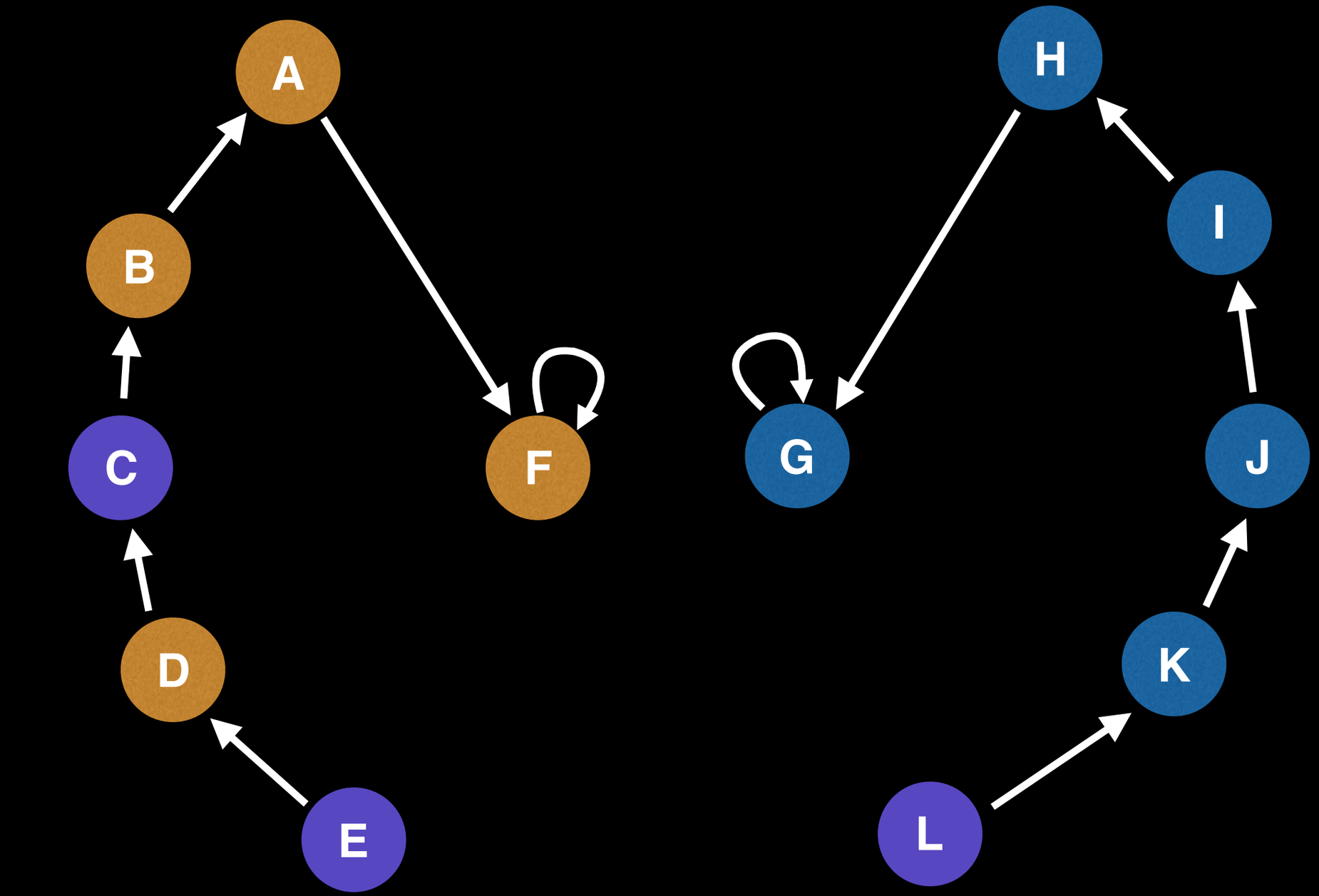
Indicates that there is a pointer to this node

Hypothetical Union Find path compression example



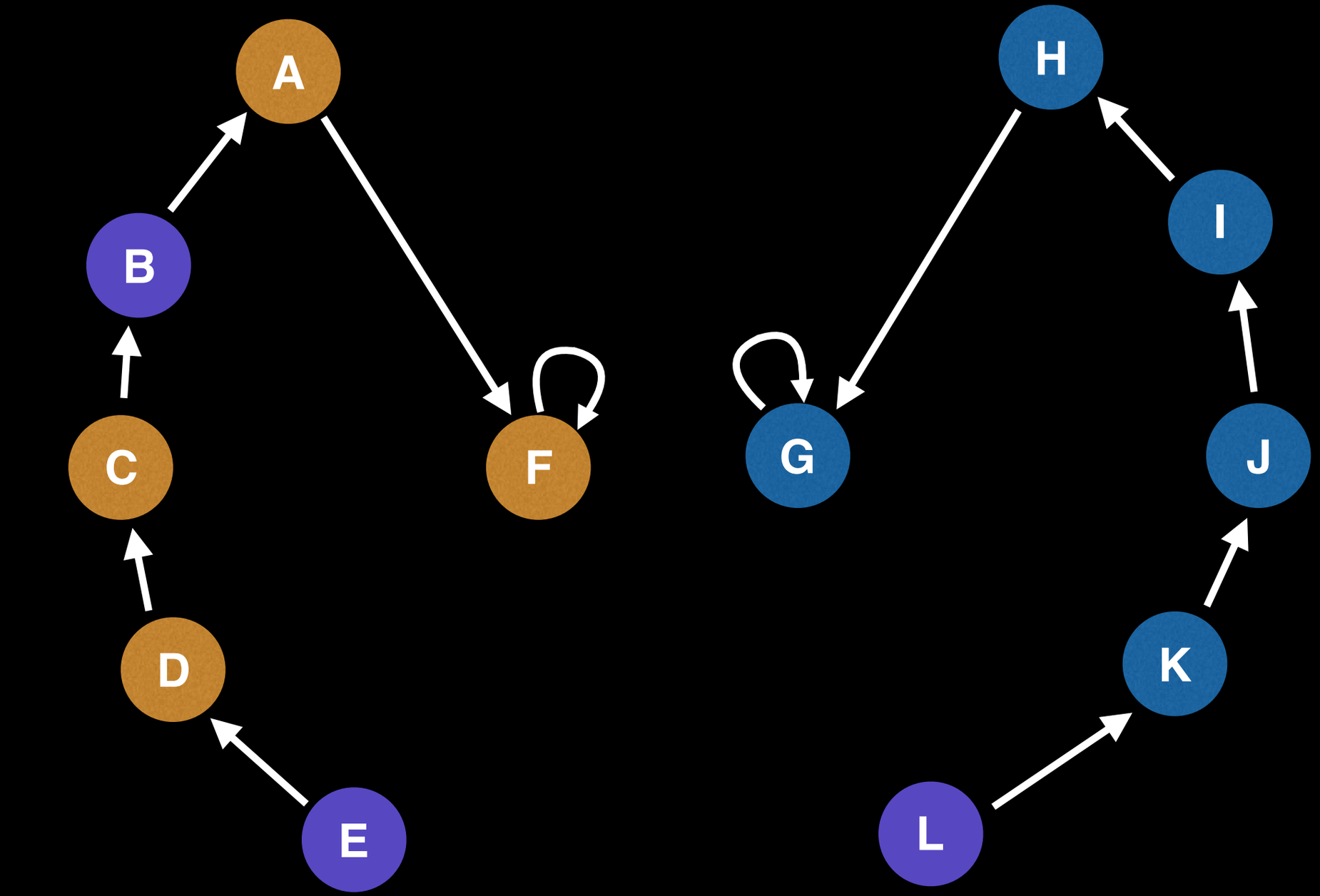
Indicates that there is a pointer to this node

Hypothetical Union Find path compression example



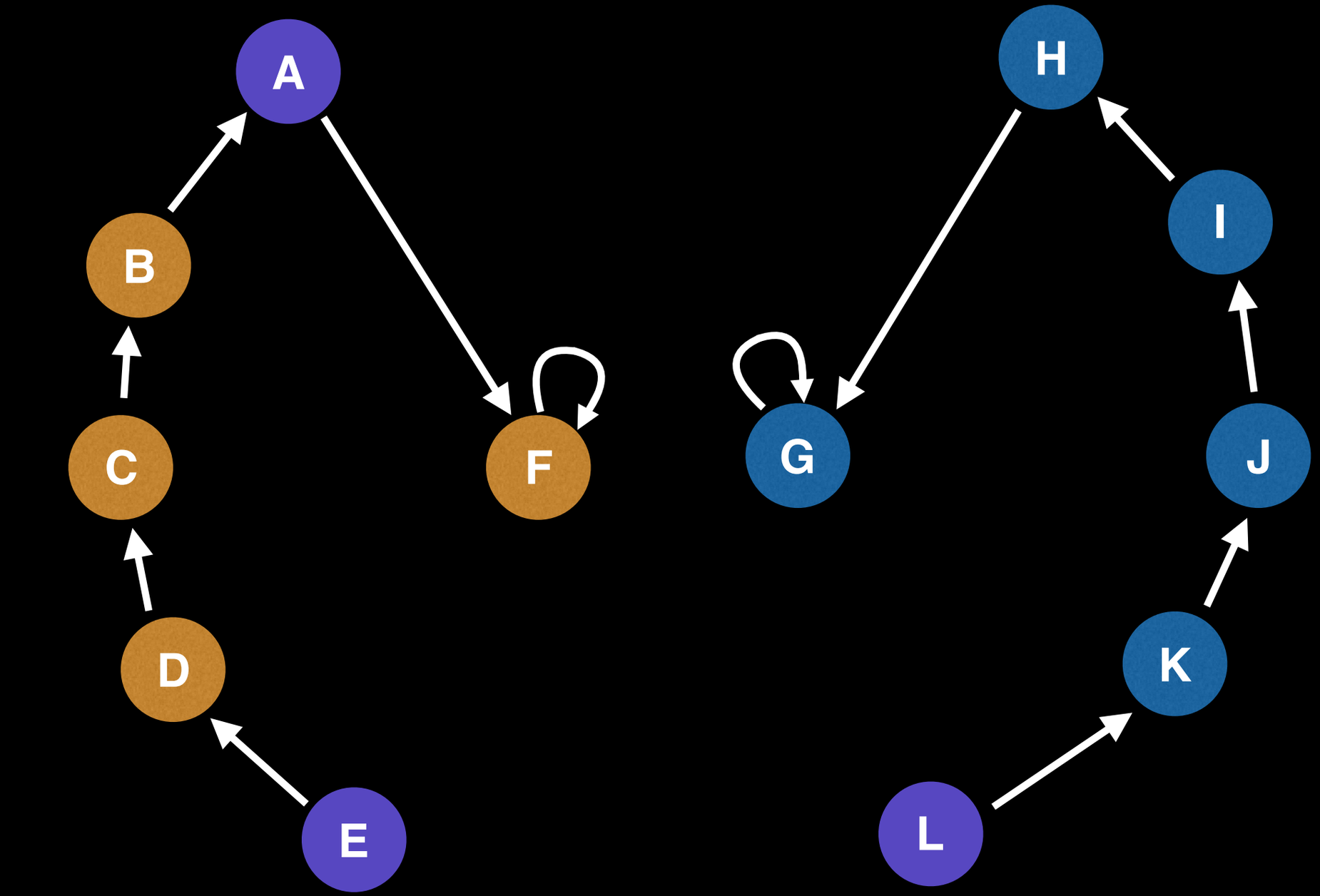
Indicates that there is a pointer to this node

Hypothetical Union Find path compression example



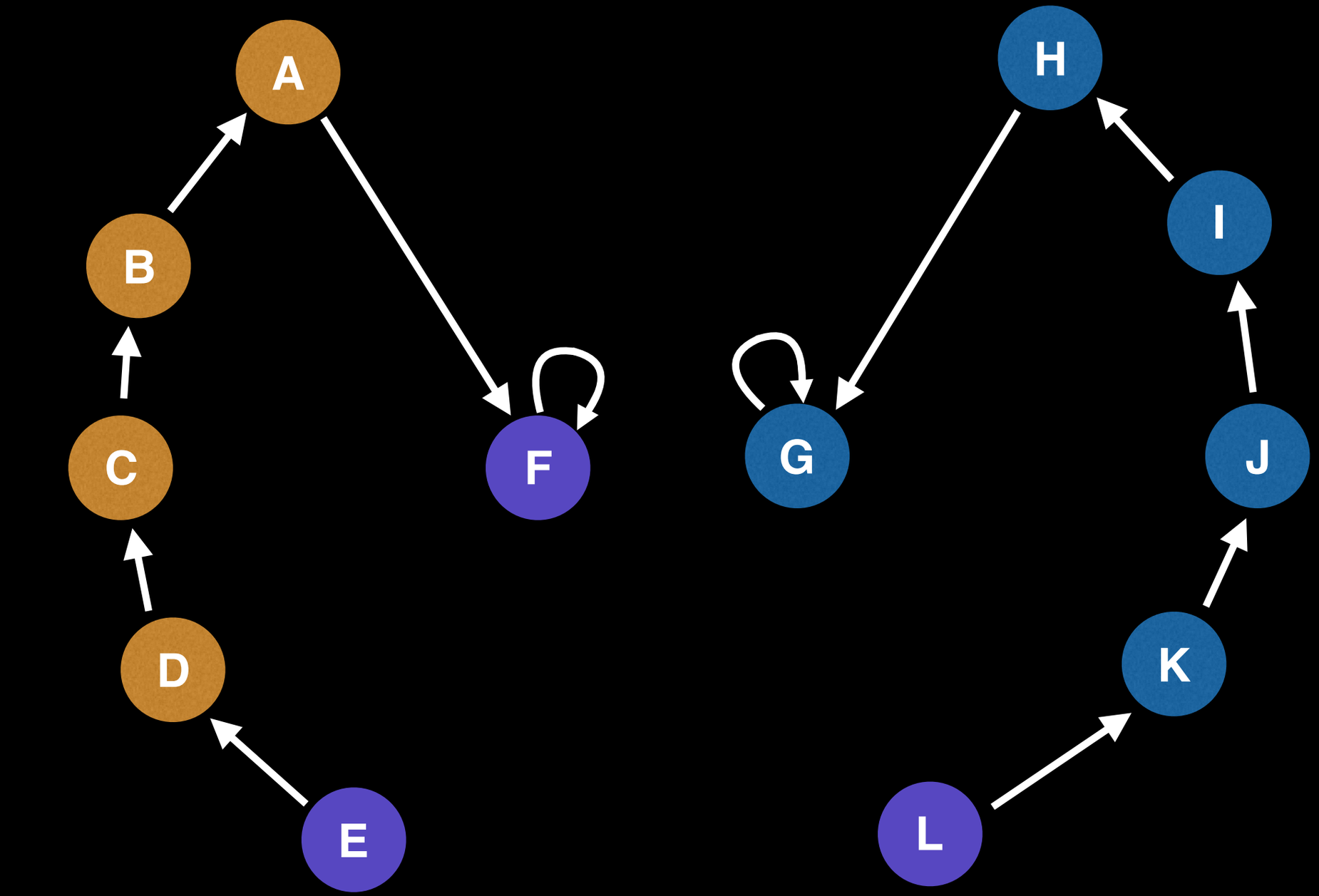
Indicates that there is a pointer to this node

Hypothetical Union Find path compression example



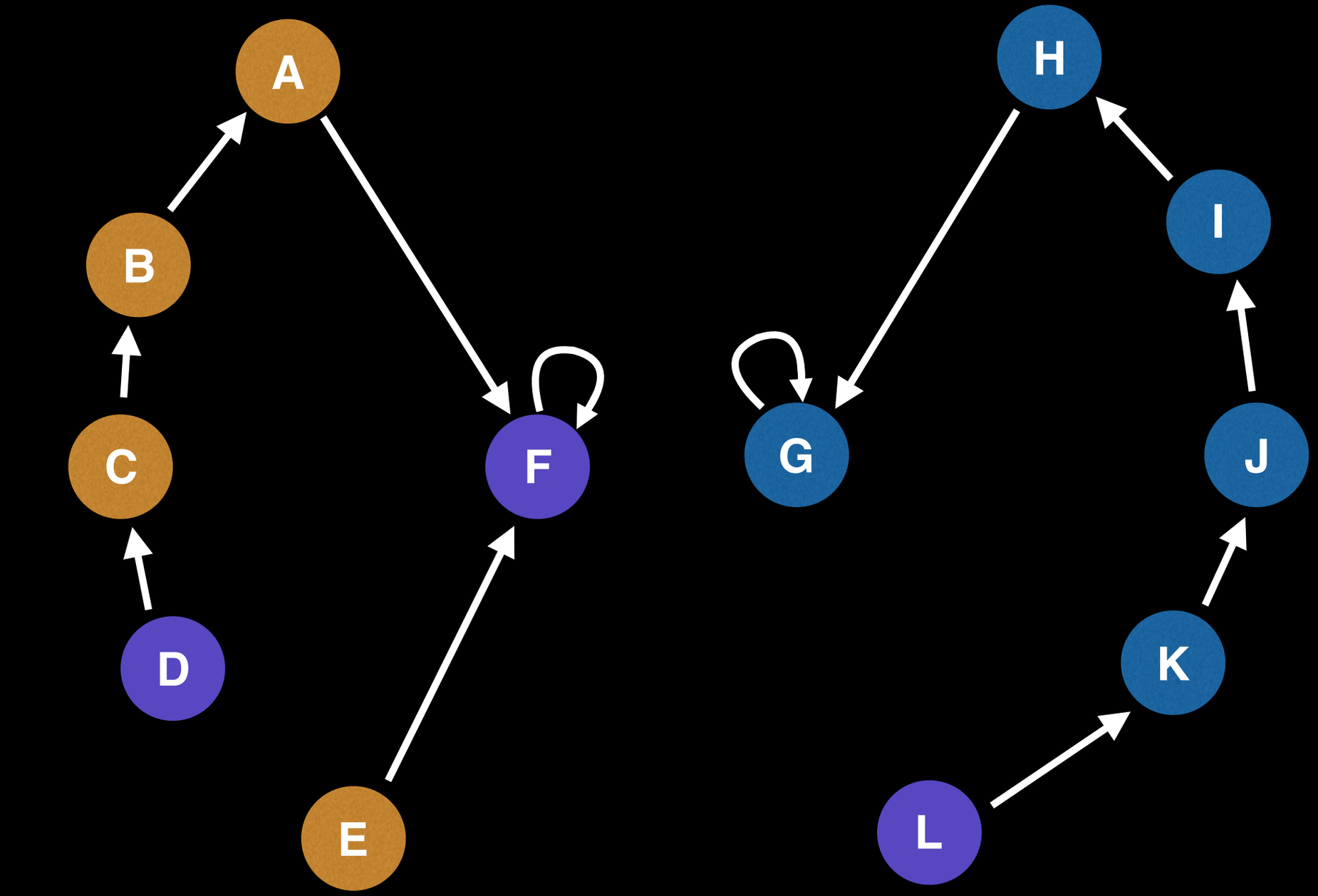
Indicates that there is a pointer to this node

Hypothetical Union Find path compression example



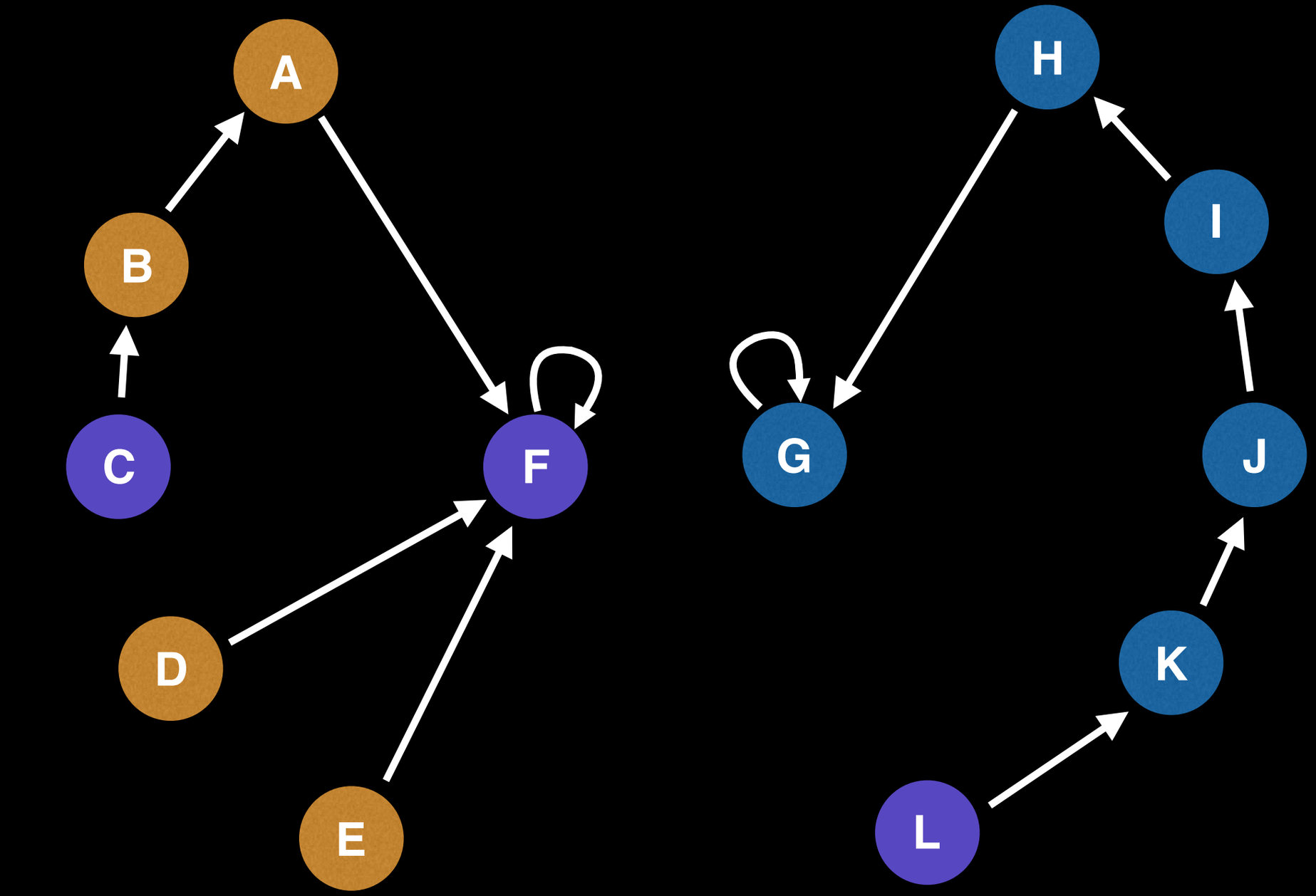
Indicates that there is a pointer to this node

Hypothetical Union Find path compression example



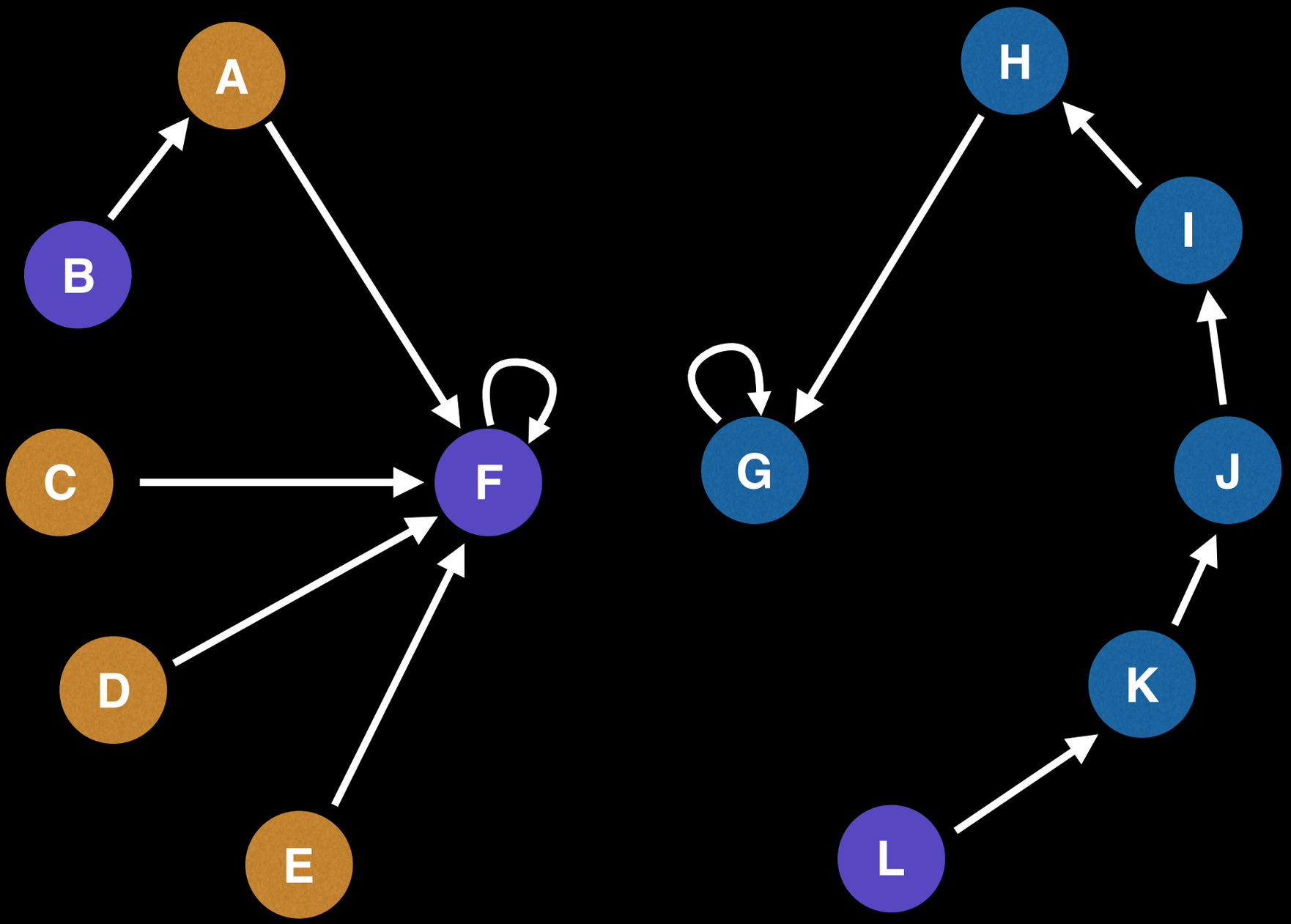
Indicates that there is a pointer to this node

Hypothetical Union Find path compression example



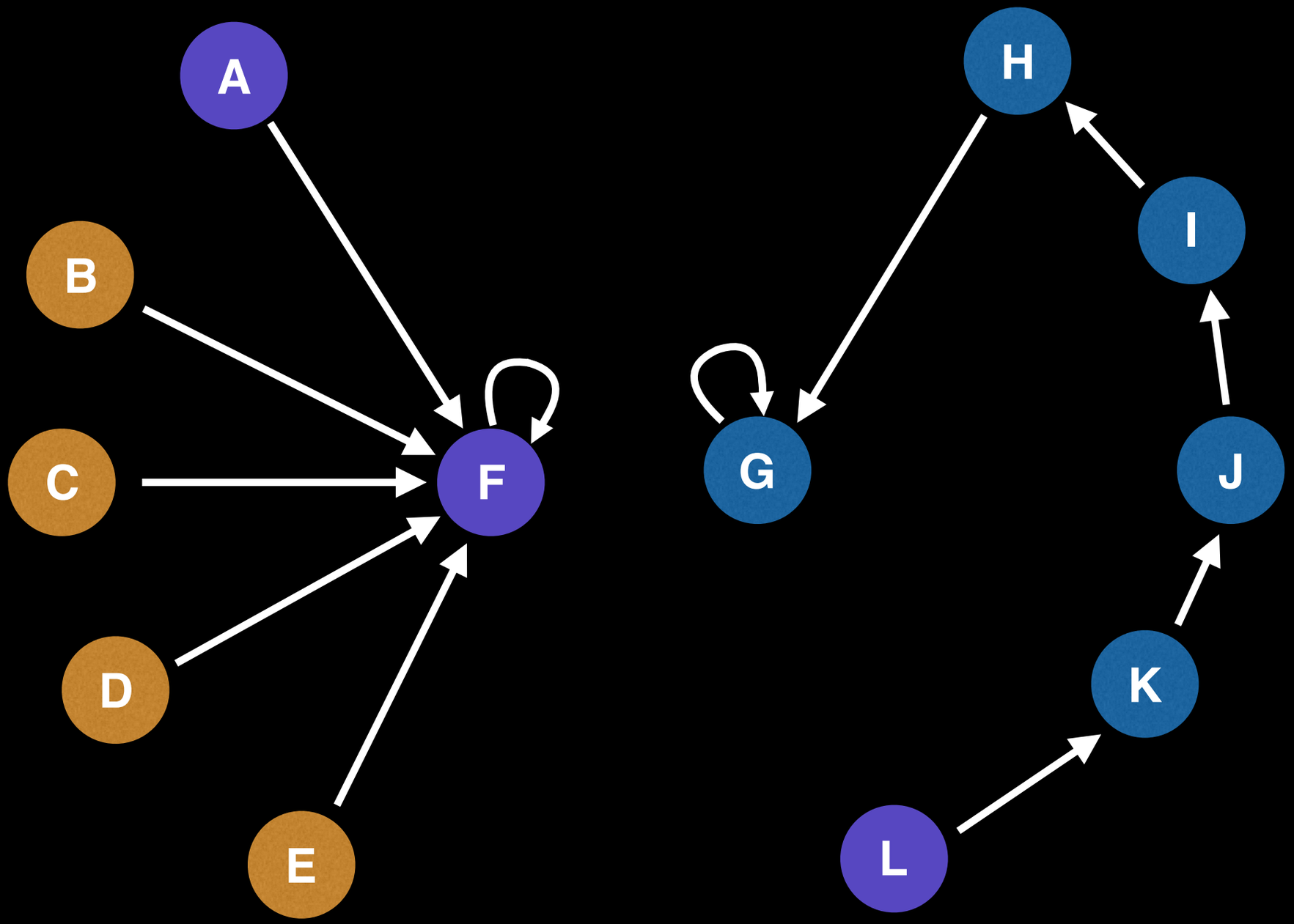
Indicates that there is a pointer to this node

Hypothetical Union Find path compression example



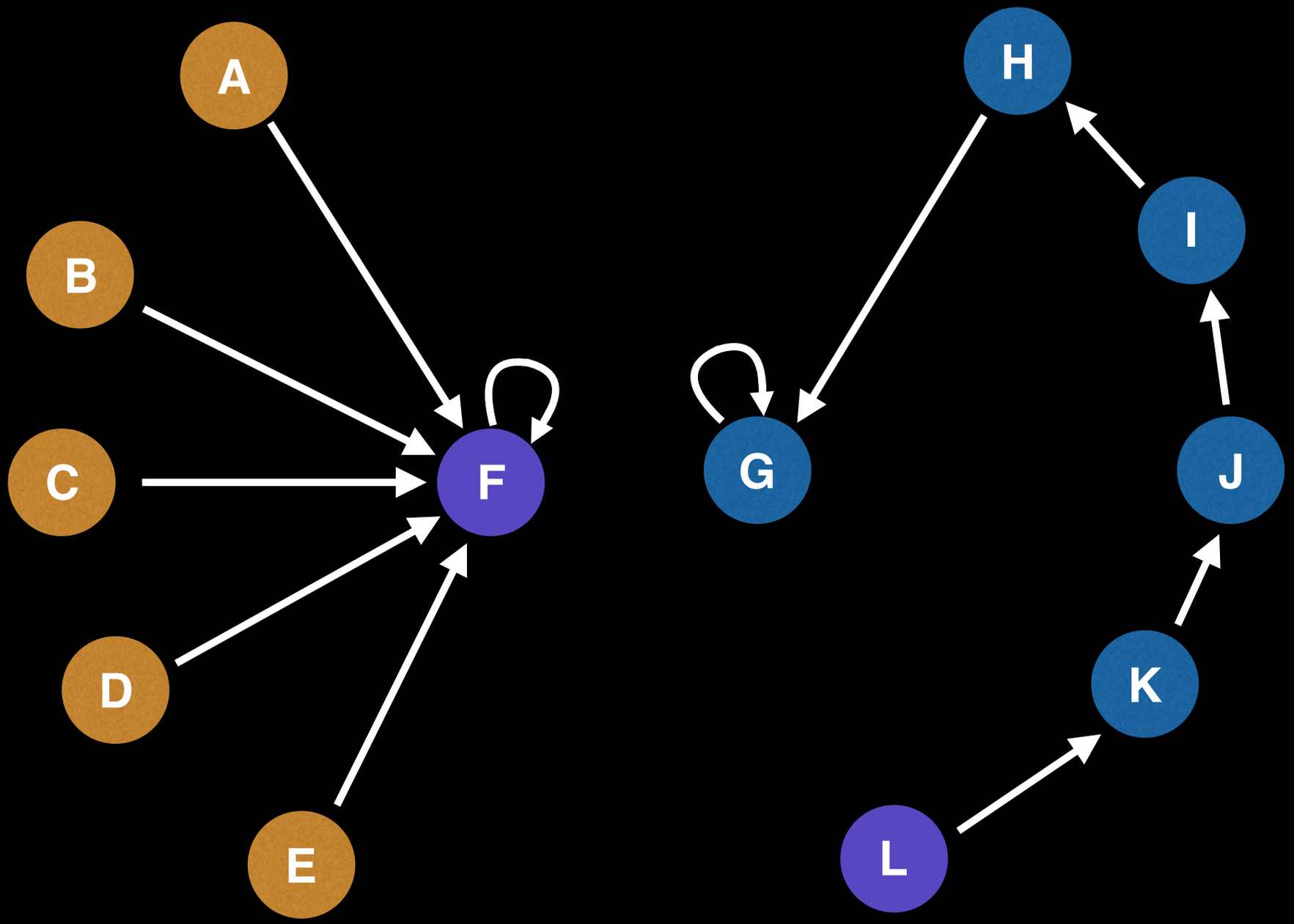
Indicates that there is a pointer to this node

Hypothetical Union Find path compression example



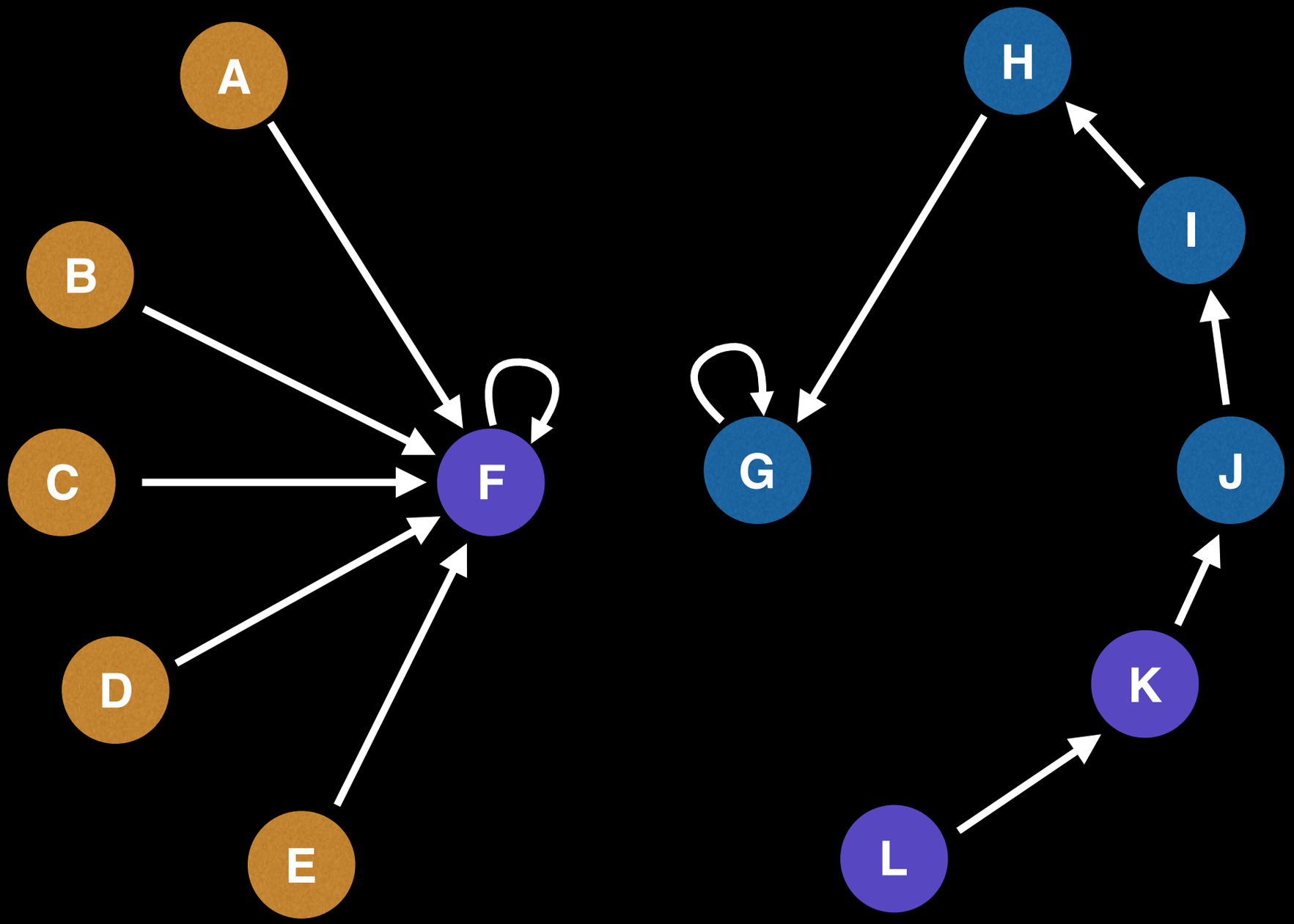
Indicates that there is a pointer to this node

Hypothetical Union Find path compression example



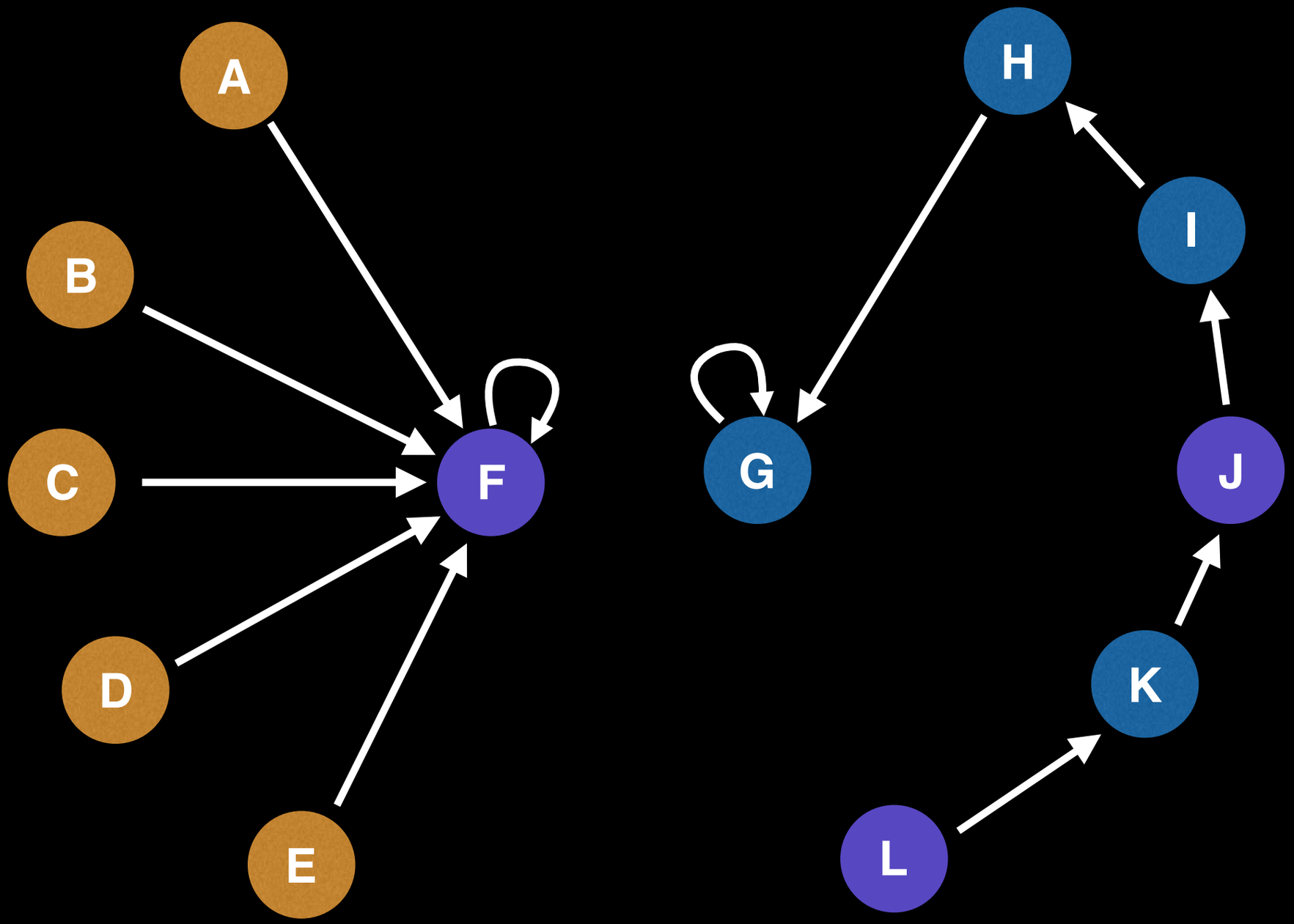
Indicates that there is a pointer to this node

Hypothetical Union Find path compression example



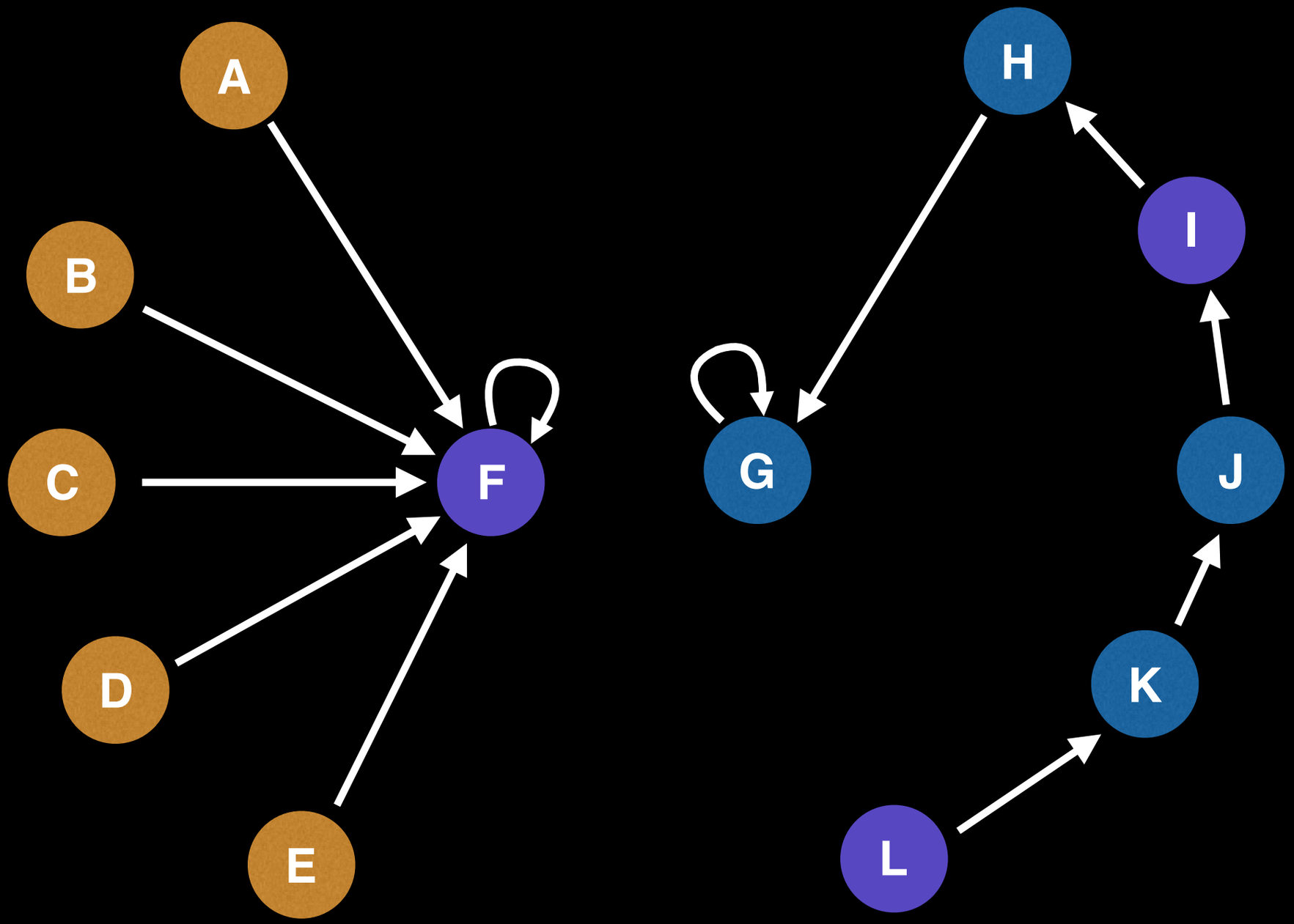
Indicates that there is a pointer to this node

Hypothetical Union Find path compression example



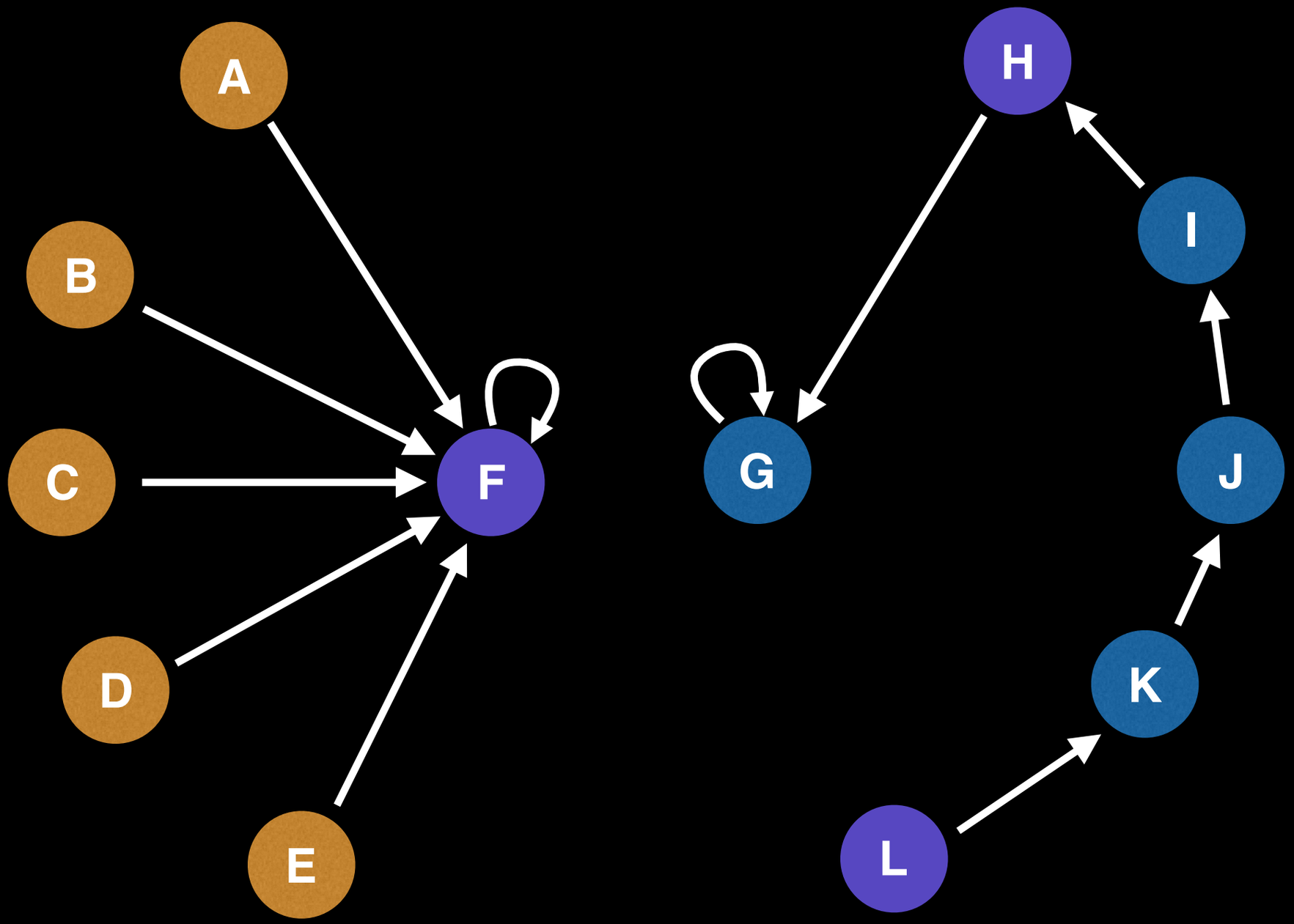
Indicates that there is a pointer to this node

Hypothetical Union Find path compression example



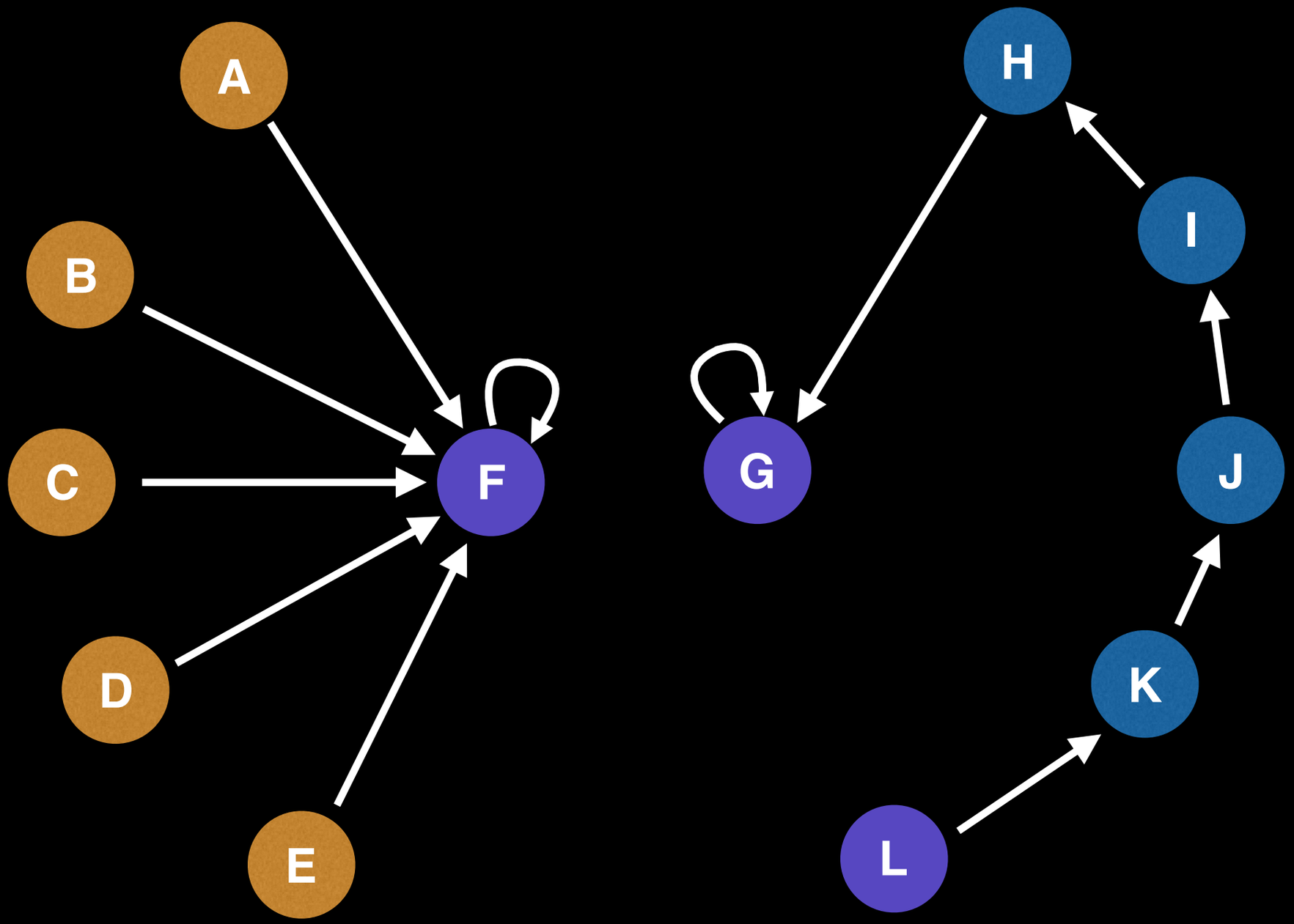
Indicates that there is a pointer to this node

Hypothetical Union Find path compression example



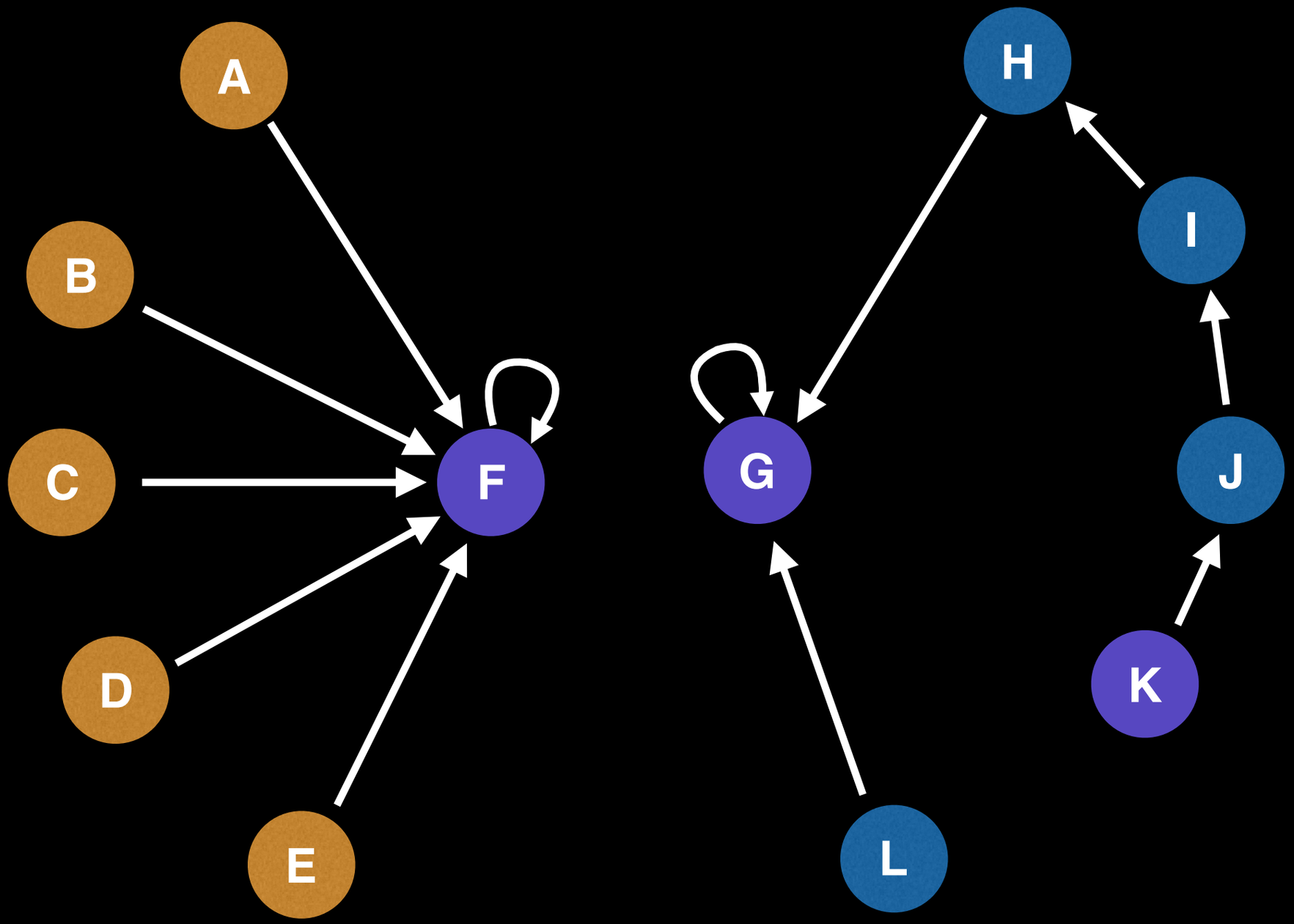
Indicates that there is a pointer to this node

Hypothetical Union Find path compression example



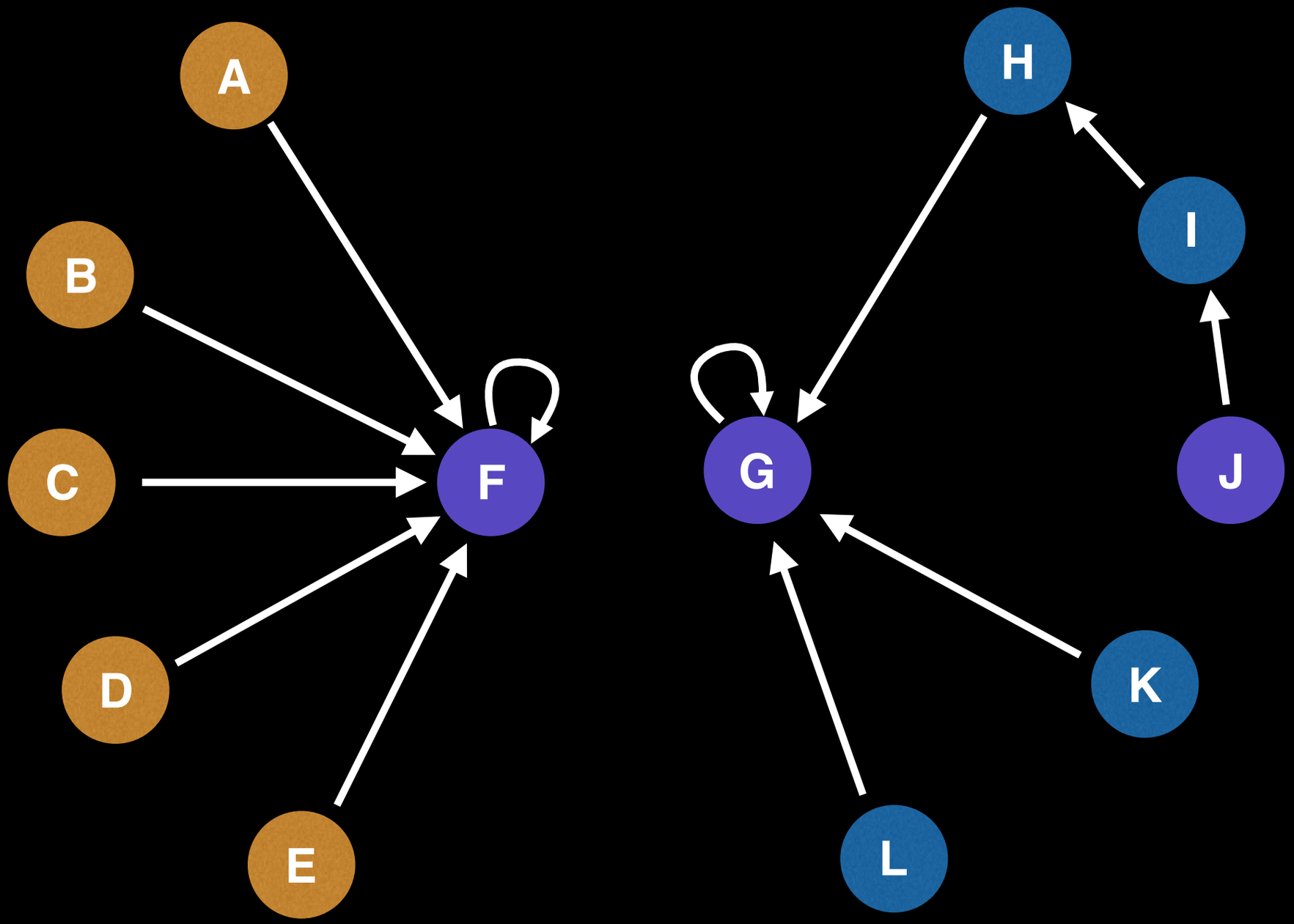
Indicates that there is a pointer to this node

Hypothetical Union Find path compression example



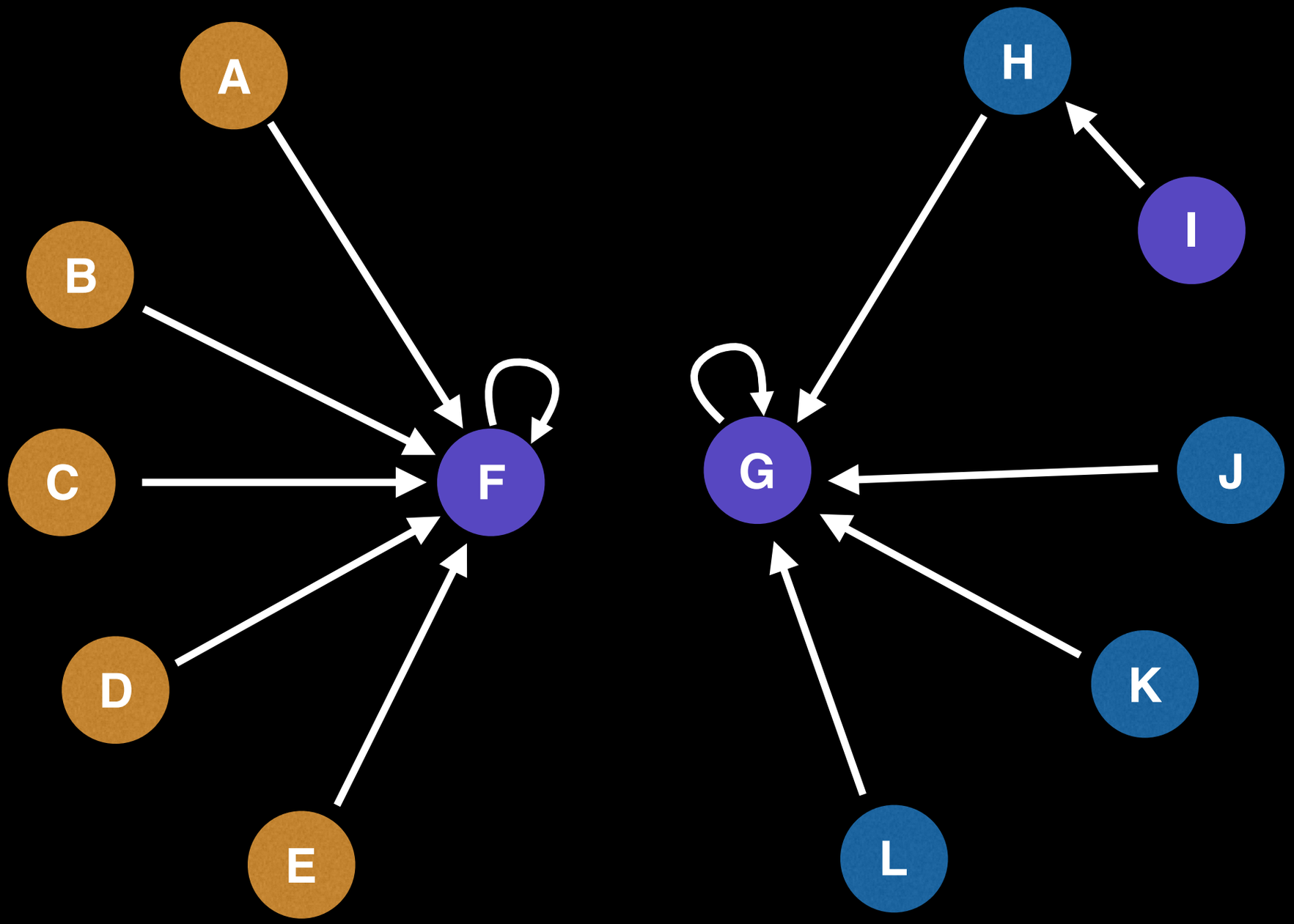
Indicates that there is a pointer to this node

Hypothetical Union Find path compression example



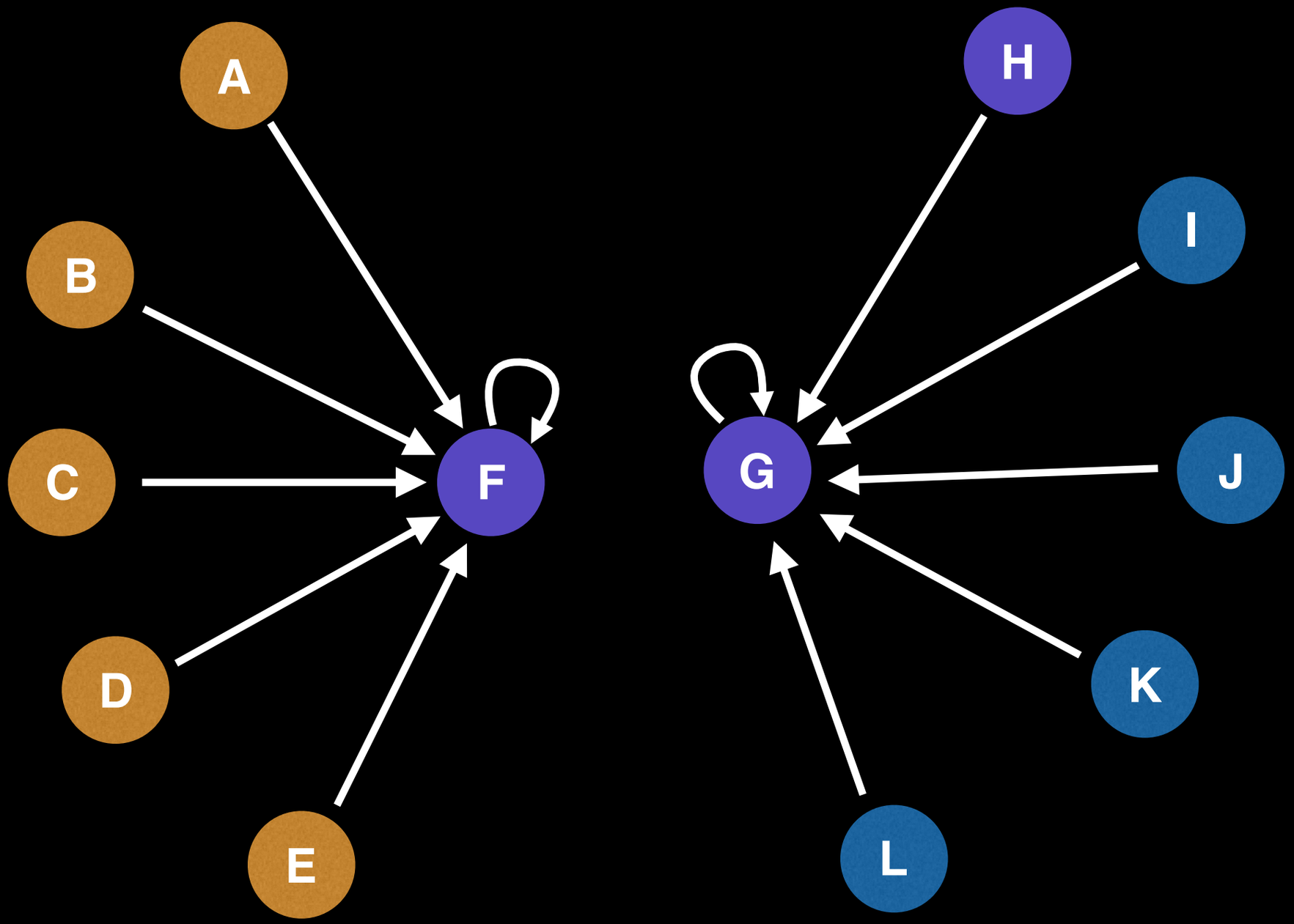
Indicates that there is a pointer to this node

Hypothetical Union Find path compression example



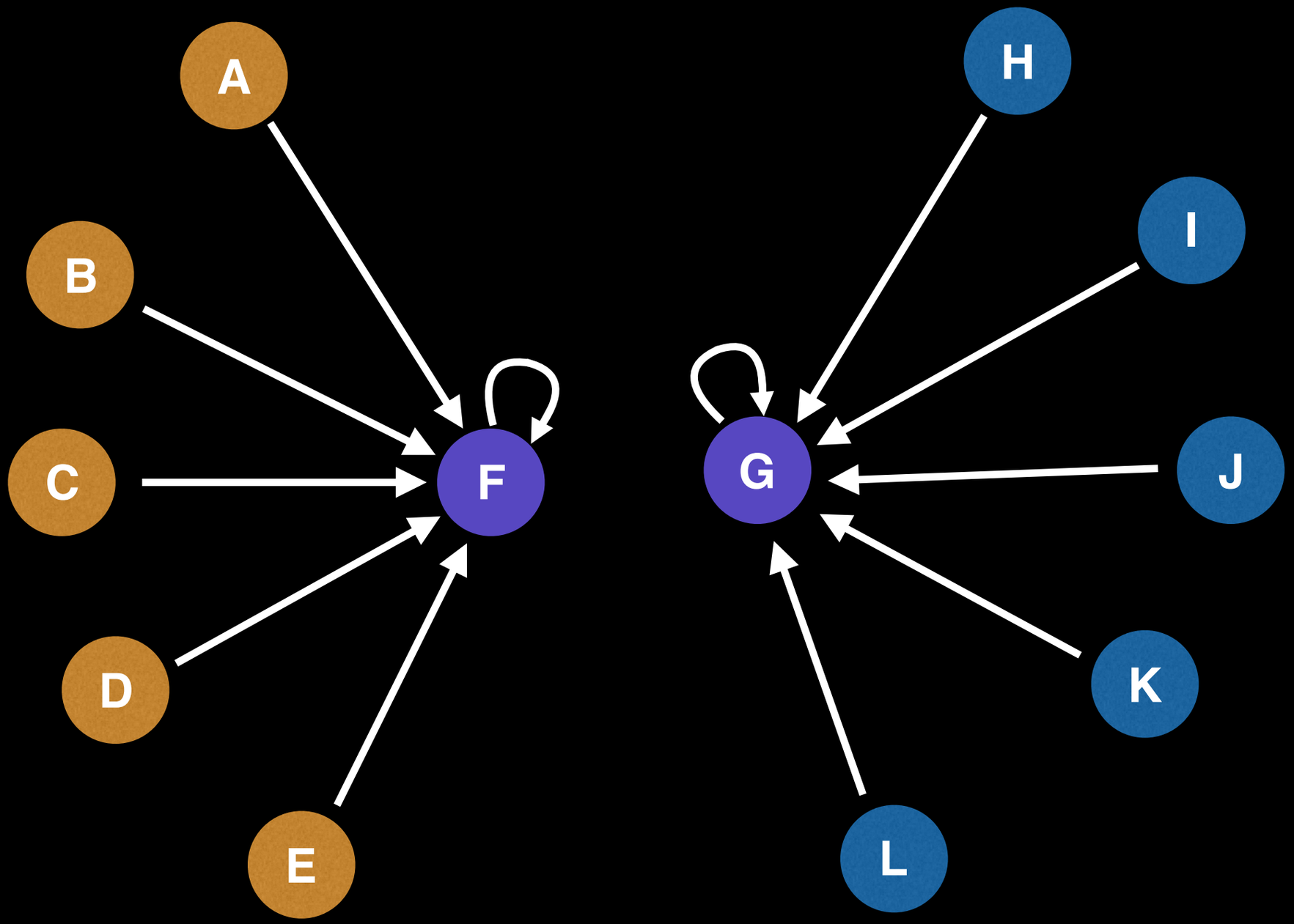
Indicates that there is a pointer to this node

Hypothetical Union Find path compression example



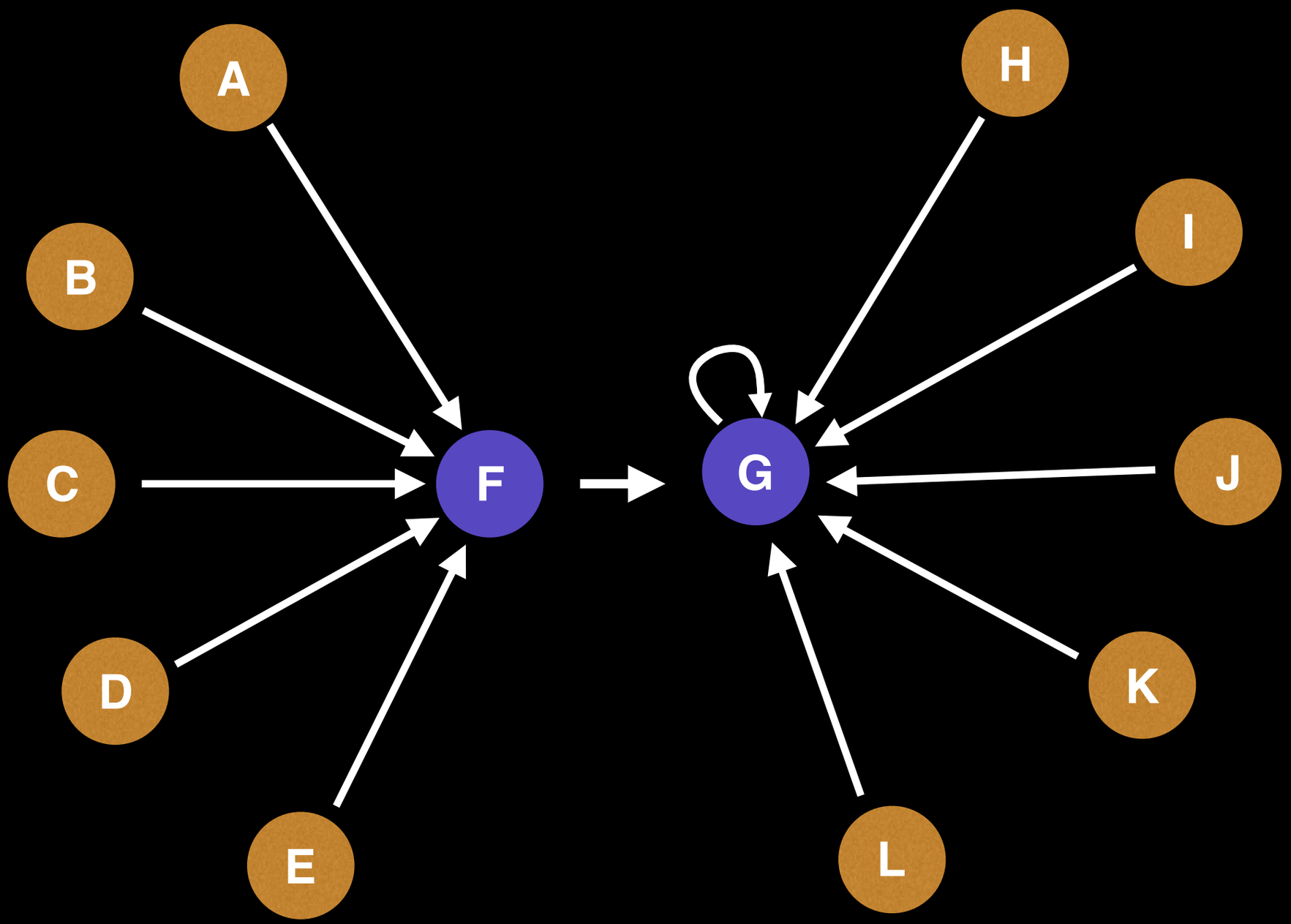
Indicates that there is a pointer to this node

Hypothetical Union Find path compression example



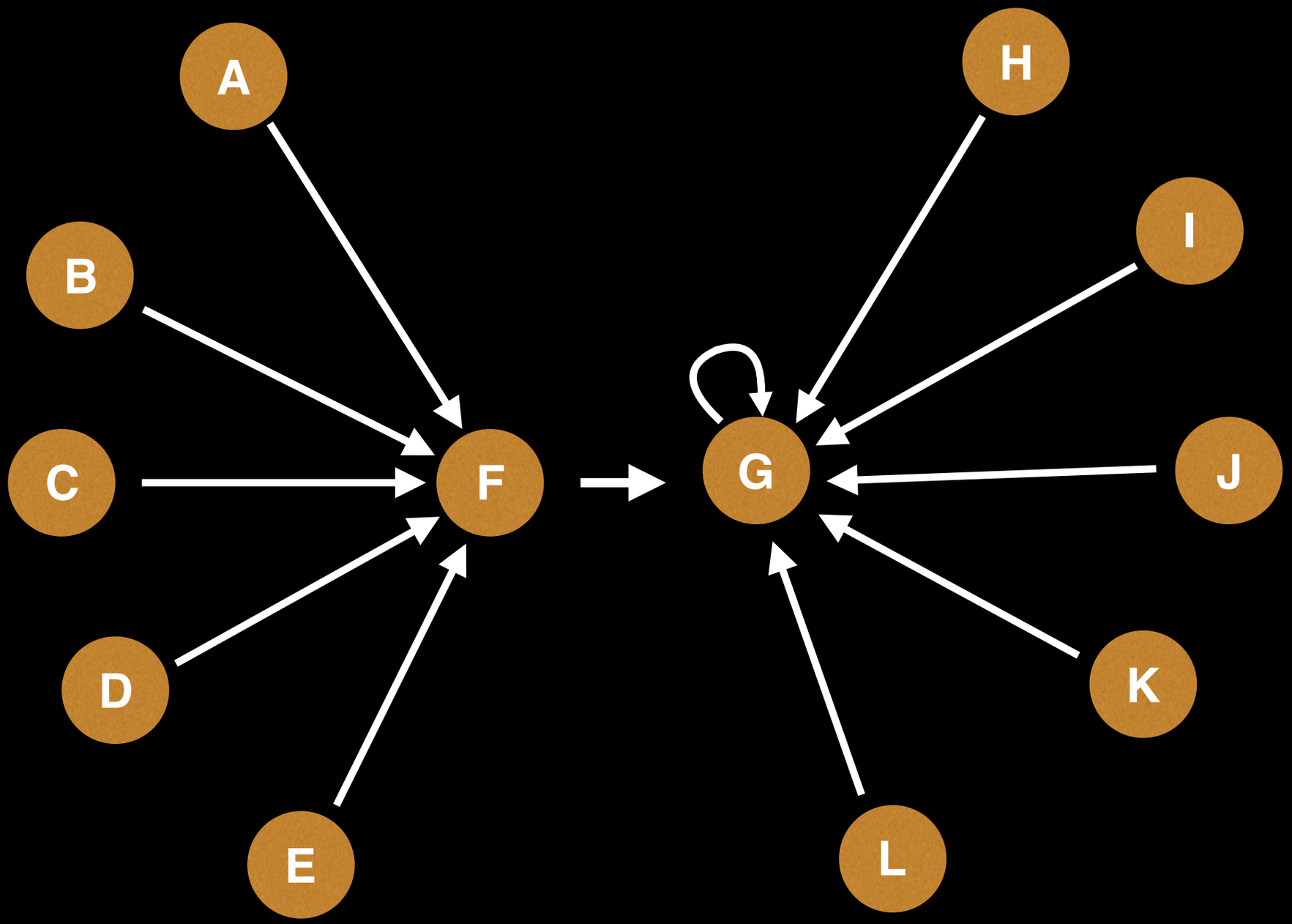
Indicates that there is a pointer to this node

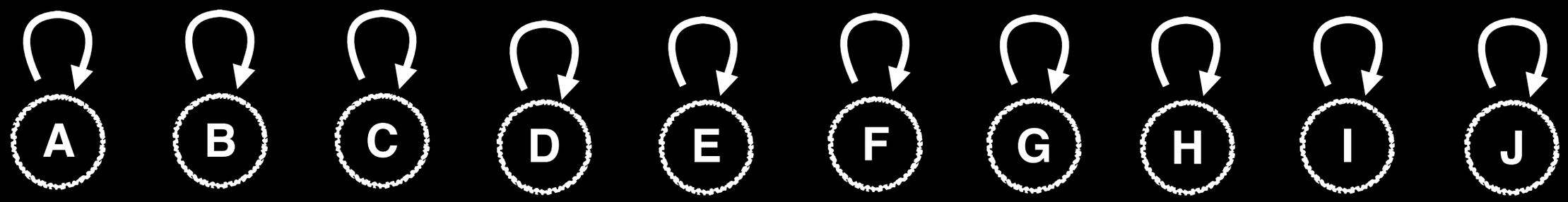
Hypothetical Union Find path compression example



Indicates that there is a pointer to this node

Hypothetical Union Find path compression example



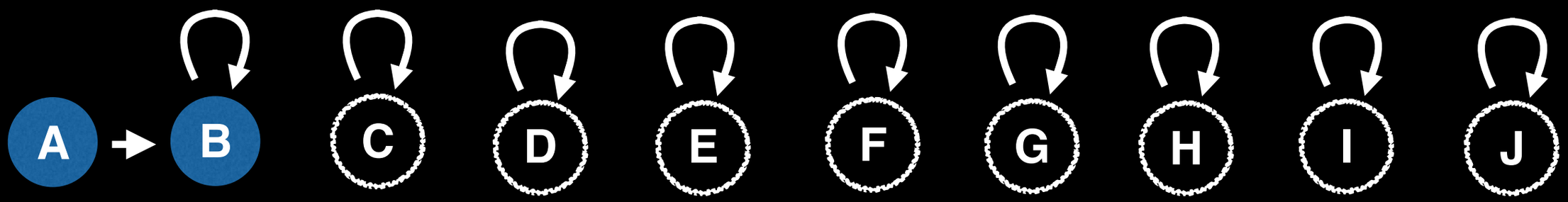


Using regular union find method

Instructions:

Union(A,B)
Union(C,D)
Union(E,F)
Union(G,H)
Union(I,J)

Union(J,G)
Union(H,F)
Union(A,C)
Union(D,E)
Union(G,B)
Union(I,J)

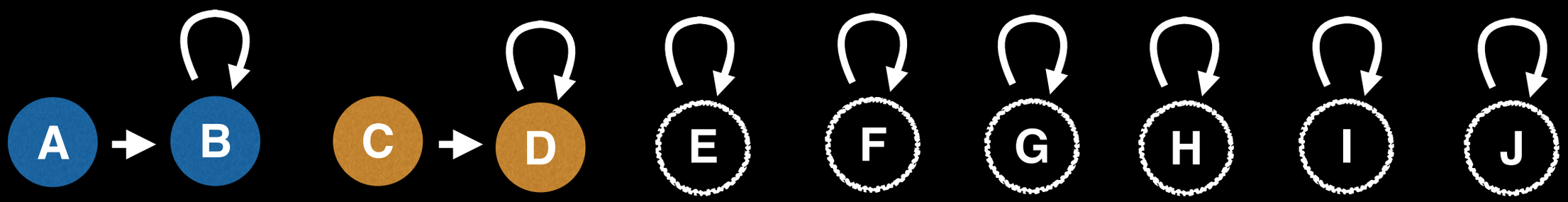


Using regular union find method

Instructions:

Union(A,B)
Union(C,D)
Union(E,F)
Union(G,H)
Union(I,J)

Union(J,G)
Union(H,F)
Union(A,C)
Union(D,E)
Union(G,B)
Union(I,J)

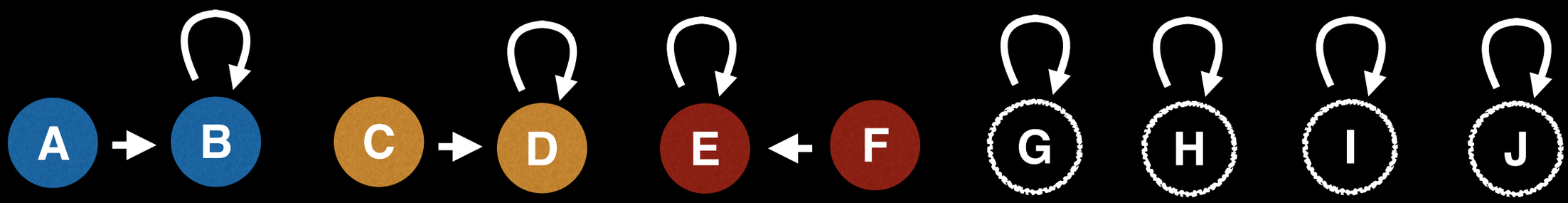


Using regular union find method

Instructions:

Union(A,B)
Union(C,D)
Union(E,F)
Union(G,H)
Union(I,J)

Union(J,G)
Union(H,F)
Union(A,C)
Union(D,E)
Union(G,B)
Union(I,J)

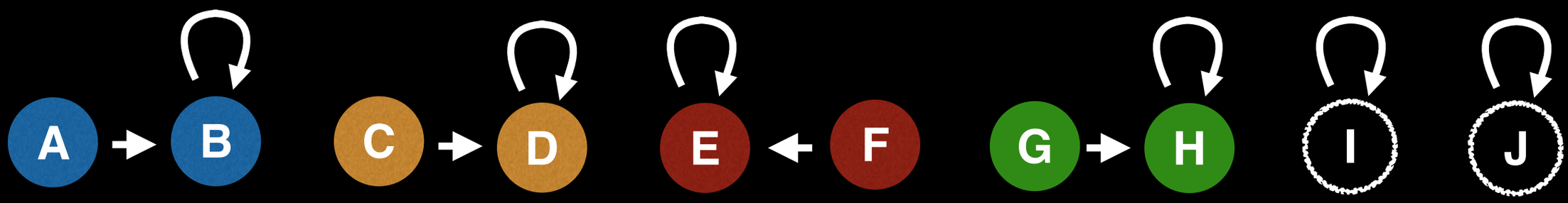


Using regular union find method

Instructions:

Union(A,B)
Union(C,D)
Union(E,F)
Union(G,H)
Union(I,J)

Union(J,G)
Union(H,F)
Union(A,C)
Union(D,E)
Union(G,B)
Union(I,J)

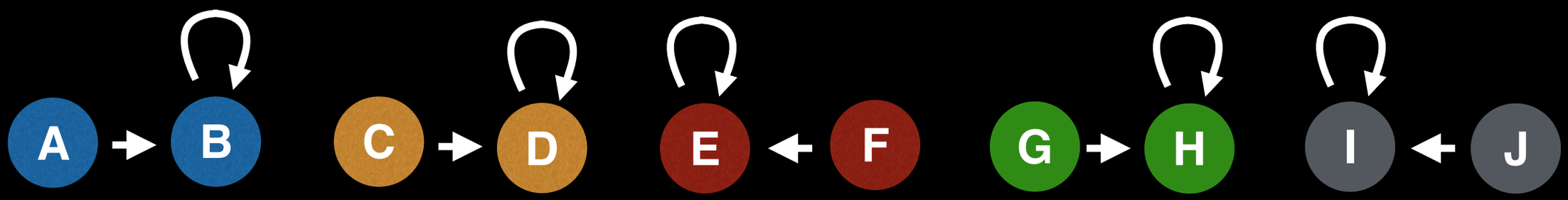


Using regular union find method

Instructions:

Union(A,B)
Union(C,D)
Union(E,F)
Union(G,H)
Union(I,J)

Union(J,G)
Union(H,F)
Union(A,C)
Union(D,E)
Union(G,B)
Union(I,J)

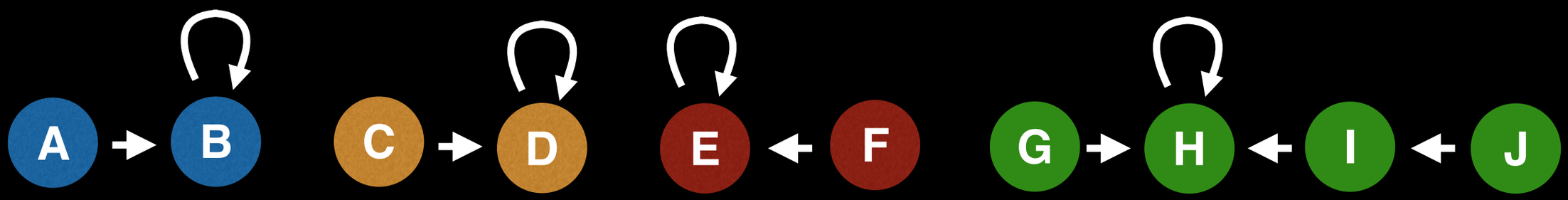


Using regular union find method

Instructions:

```
Union(A,B)
Union(C,D)
Union(E,F)
Union(G,H)
Union(I,J)
```

```
Union(J,G)
Union(H,F)
Union(A,C)
Union(D,E)
Union(G,B)
Union(I,J)
```

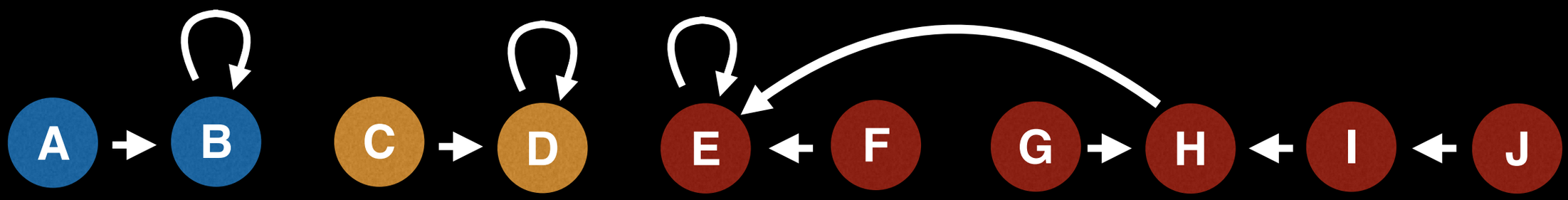


Using regular union find method

Instructions:

```
Union(A,B)
Union(C,D)
Union(E,F)
Union(G,H)
Union(I,J)
```

```
Union(J,G)
Union(H,F)
Union(A,C)
Union(D,E)
Union(G,B)
Union(I,J)
```

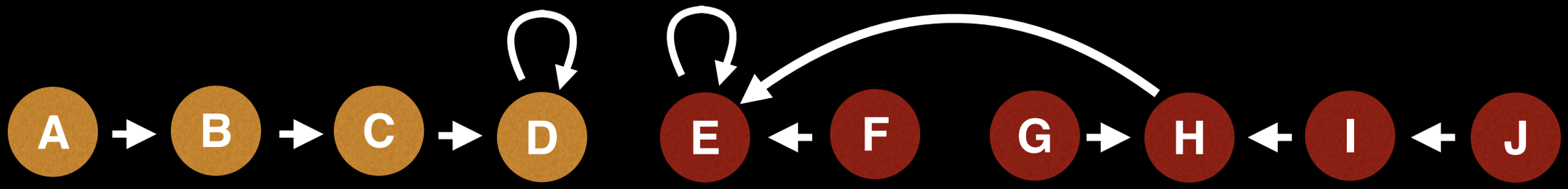


Using regular union find method

Instructions:

Union(A,B)
Union(C,D)
Union(E,F)
Union(G,H)
Union(I,J)

Union(J,G)
Union(H,F)
Union(A,C)
Union(D,E)
Union(G,B)
Union(I,J)

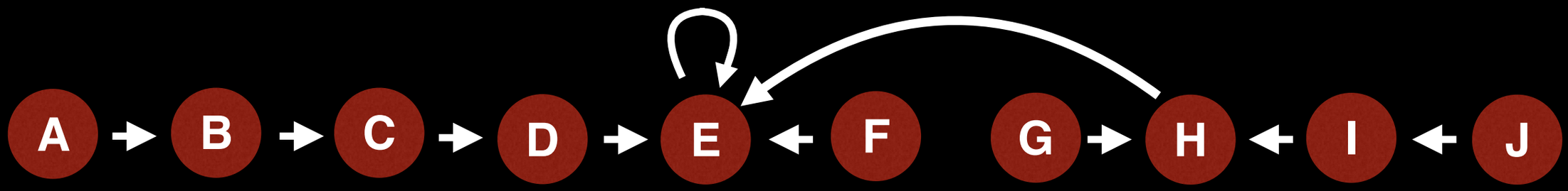


Using regular union find method

Instructions:

```
Union(A,B)
Union(C,D)
Union(E,F)
Union(G,H)
Union(I,J)
```

```
Union(J,G)
Union(H,F)
Union(A,C)
Union(D,E)
Union(G,B)
Union(I,J)
```

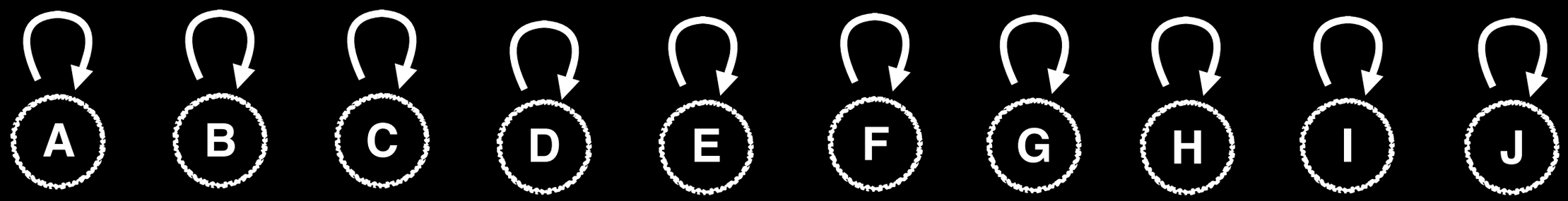


Using regular union find method

Instructions:

```
Union(A,B)
Union(C,D)
Union(E,F)
Union(G,H)
Union(I,J)
```

```
Union(J,G)
Union(H,F)
Union(A,C)
Union(D,E)
Union(G,B)
Union(I,J)
```

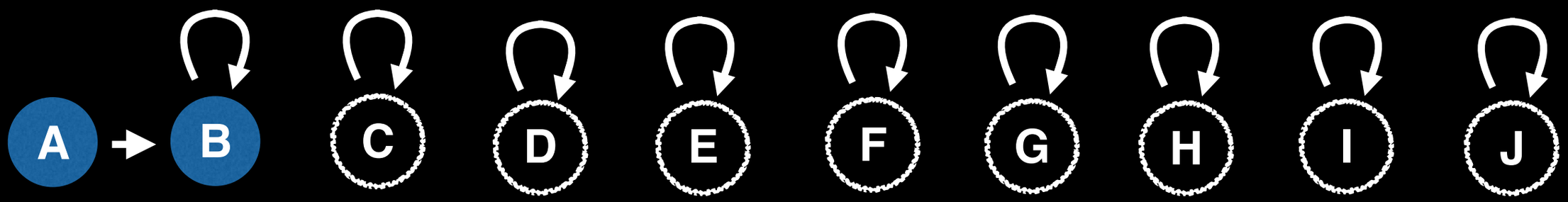


Using **path compression**

Instructions:

```
Union(A,B)
Union(C,D)
Union(E,F)
Union(G,H)
Union(I,J)
```

```
Union(J,G)
Union(H,F)
Union(A,C)
Union(D,E)
Union(G,B)
Union(I,J)
```

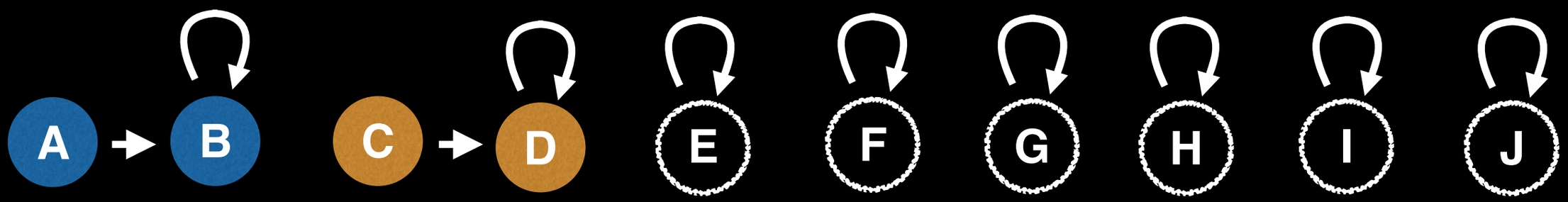


Using **path compression**

Instructions:

```
Union(A,B)
Union(C,D)
Union(E,F)
Union(G,H)
Union(I,J)
```

```
Union(J,G)
Union(H,F)
Union(A,C)
Union(D,E)
Union(G,B)
Union(I,J)
```

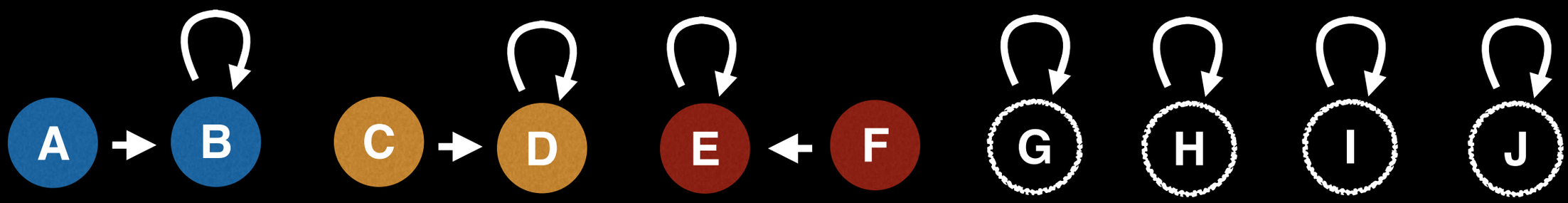


Using **path compression**

Instructions:

```
Union(A,B)
Union(C,D)
Union(E,F)
Union(G,H)
Union(I,J)
```

```
Union(J,G)
Union(H,F)
Union(A,C)
Union(D,E)
Union(G,B)
Union(I,J)
```

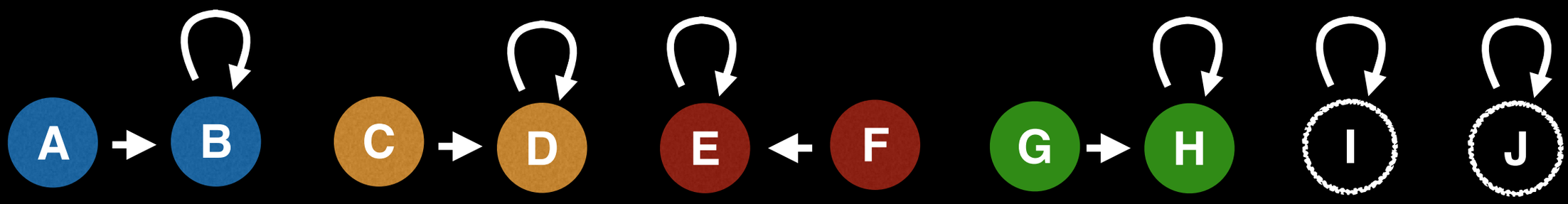


Using **path compression**

Instructions:

```
Union(A,B)
Union(C,D)
Union(E,F)
Union(G,H)
Union(I,J)
```

```
Union(J,G)
Union(H,F)
Union(A,C)
Union(D,E)
Union(G,B)
Union(I,J)
```

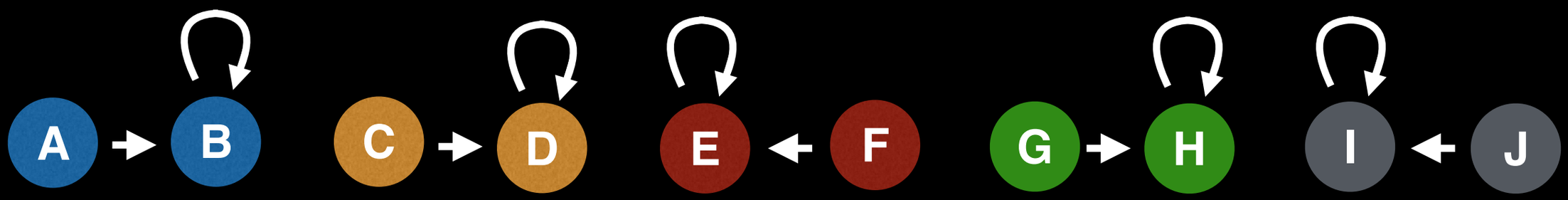


Using **path compression**

Instructions:

```
Union(A,B)
Union(C,D)
Union(E,F)
Union(G,H)
Union(I,J)
```

```
Union(J,G)
Union(H,F)
Union(A,C)
Union(D,E)
Union(G,B)
Union(I,J)
```

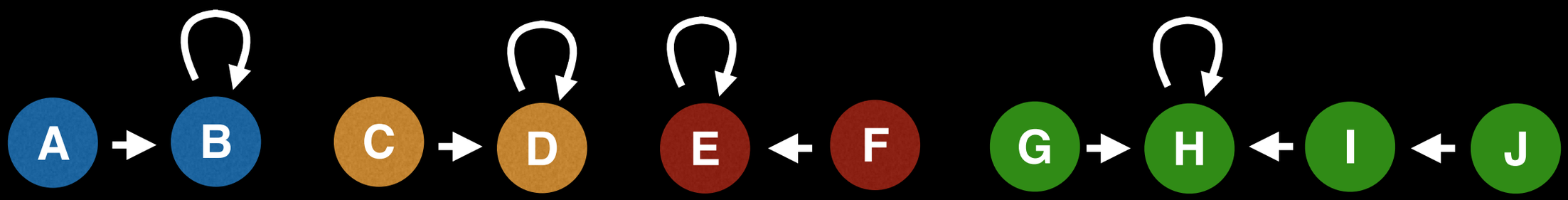


Using **path compression**

Instructions:

```
Union(A,B)
Union(C,D)
Union(E,F)
Union(G,H)
Union(I,J)
```

```
Union(J,G)
Union(H,F)
Union(A,C)
Union(D,E)
Union(G,B)
Union(I,J)
```

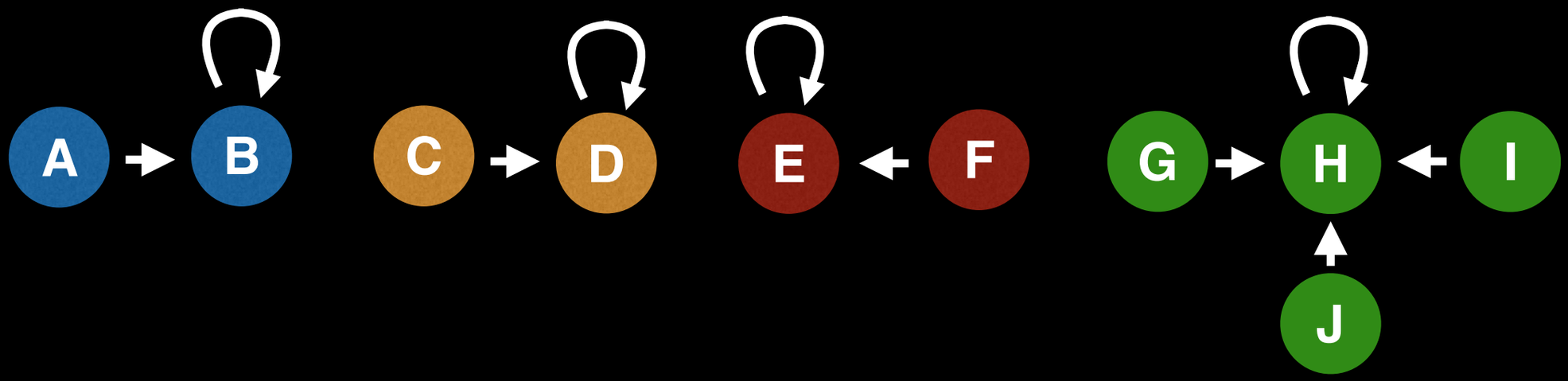


Using **path compression**

Instructions:

```
Union(A,B)
Union(C,D)
Union(E,F)
Union(G,H)
Union(I,J)
```

```
Union(J,G)
Union(H,F)
Union(A,C)
Union(D,E)
Union(G,B)
Union(I,J)
```

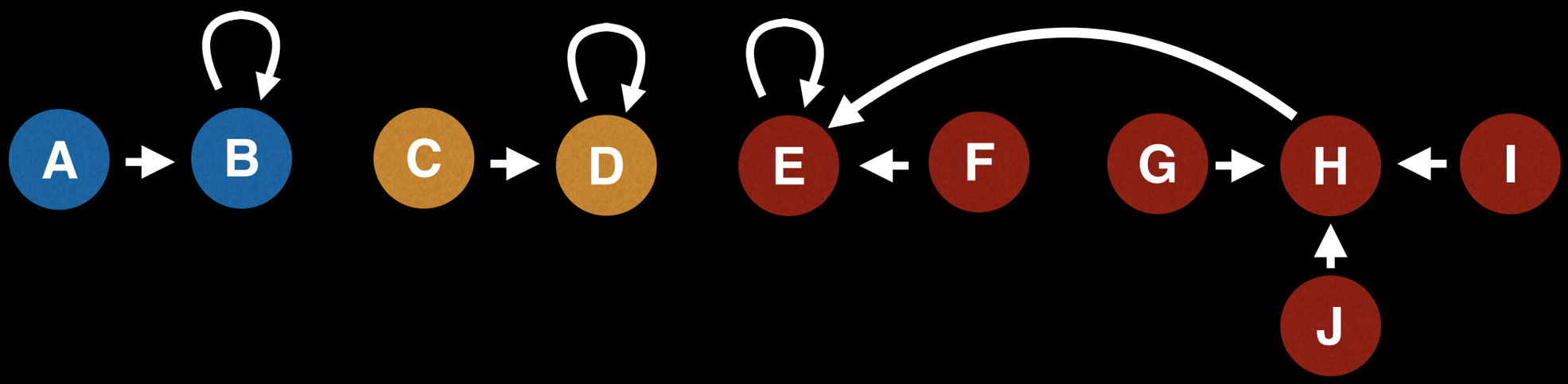


Using **path compression**

Instructions:

```
Union(A,B)
Union(C,D)
Union(E,F)
Union(G,H)
Union(I,J)
```

```
Union(J,G)
Union(H,F)
Union(A,C)
Union(D,E)
Union(G,B)
Union(I,J)
```

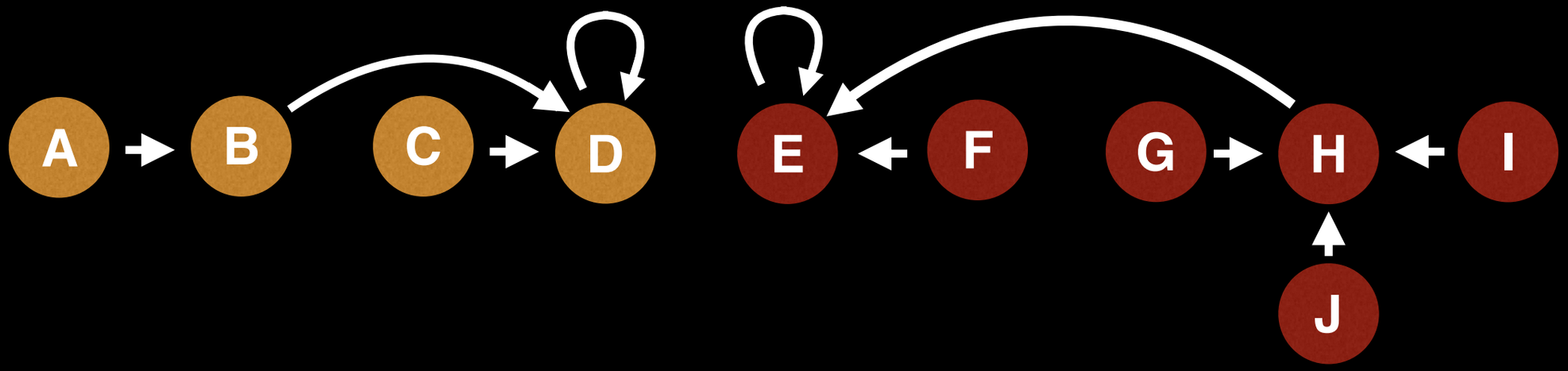


Using **path compression**

Instructions:

```
Union(A,B)
Union(C,D)
Union(E,F)
Union(G,H)
Union(I,J)
```

```
Union(J,G)
Union(H,F)
Union(A,C)
Union(D,E)
Union(G,B)
Union(I,J)
```

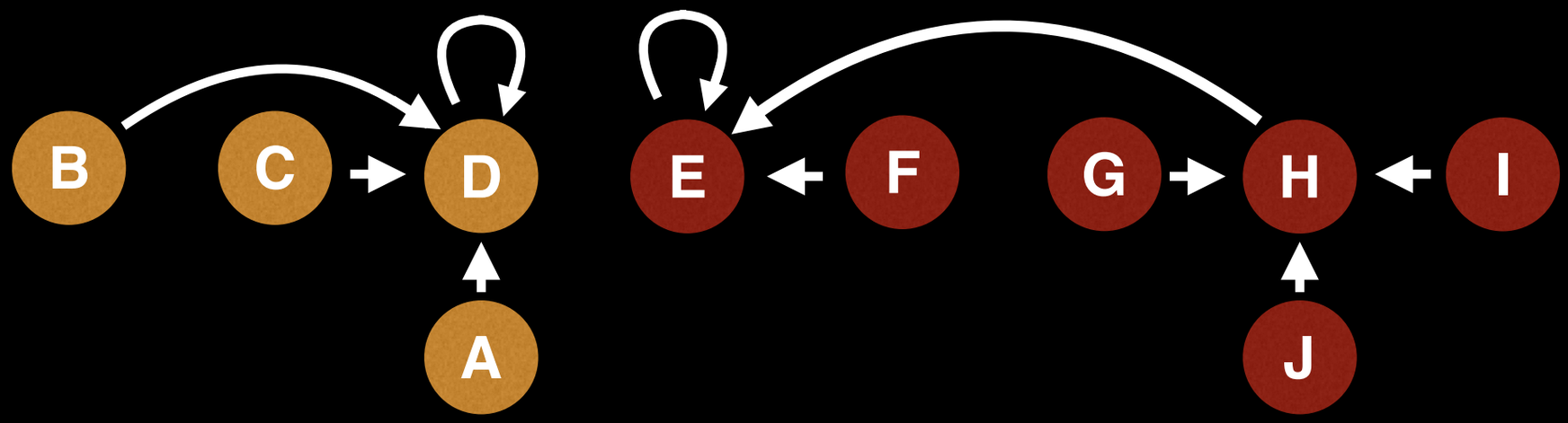


Using **path compression**

Instructions:

```
Union(A,B)
Union(C,D)
Union(E,F)
Union(G,H)
Union(I,J)
```

```
Union(J,G)
Union(H,F)
Union(A,C)
Union(D,E)
Union(G,B)
Union(I,J)
```

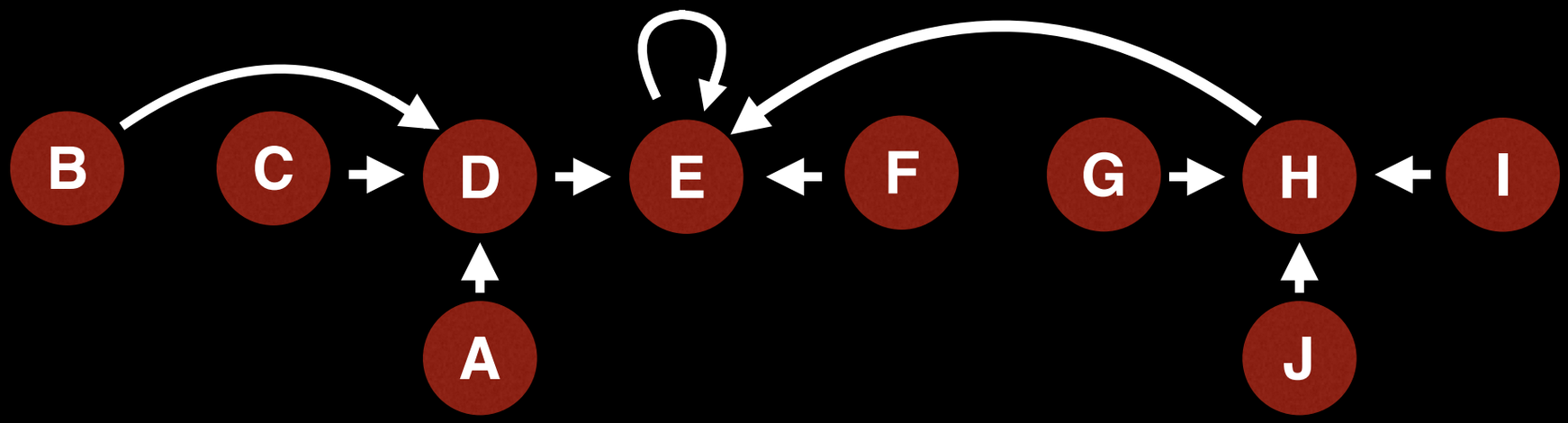


Using **path compression**

Instructions:

```
Union(A,B)
Union(C,D)
Union(E,F)
Union(G,H)
Union(I,J)
```

```
Union(J,G)
Union(H,F)
Union(A,C)
Union(D,E)
Union(G,B)
Union(I,J)
```

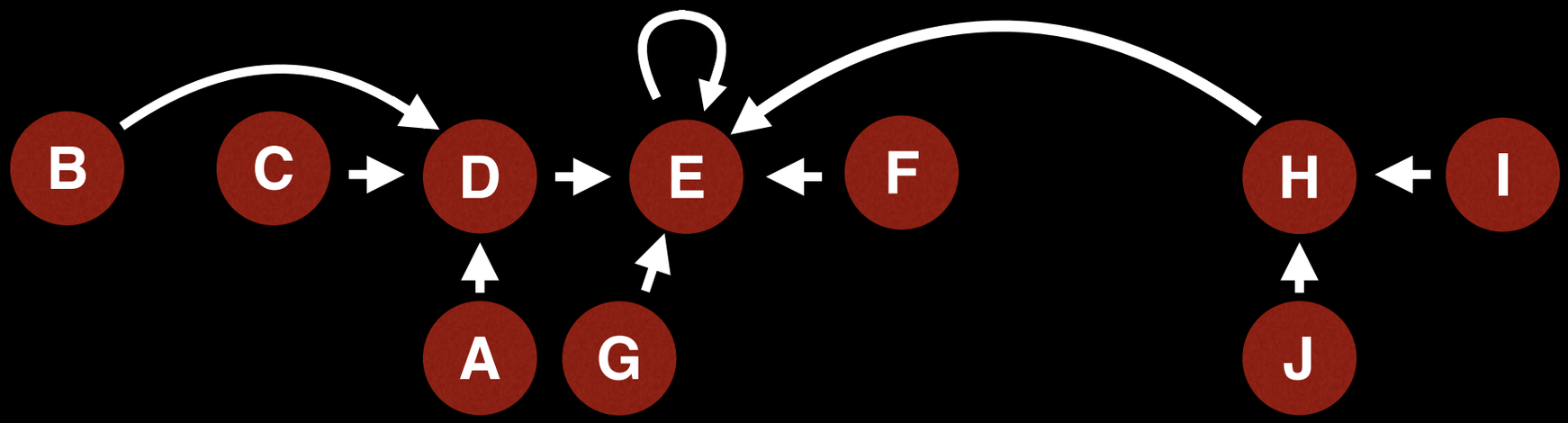


Using **path compression**

Instructions:

```
Union(A,B)
Union(C,D)
Union(E,F)
Union(G,H)
Union(I,J)
```

```
Union(J,G)
Union(H,F)
Union(A,C)
Union(D,E)
Union(G,B)
Union(I,J)
```

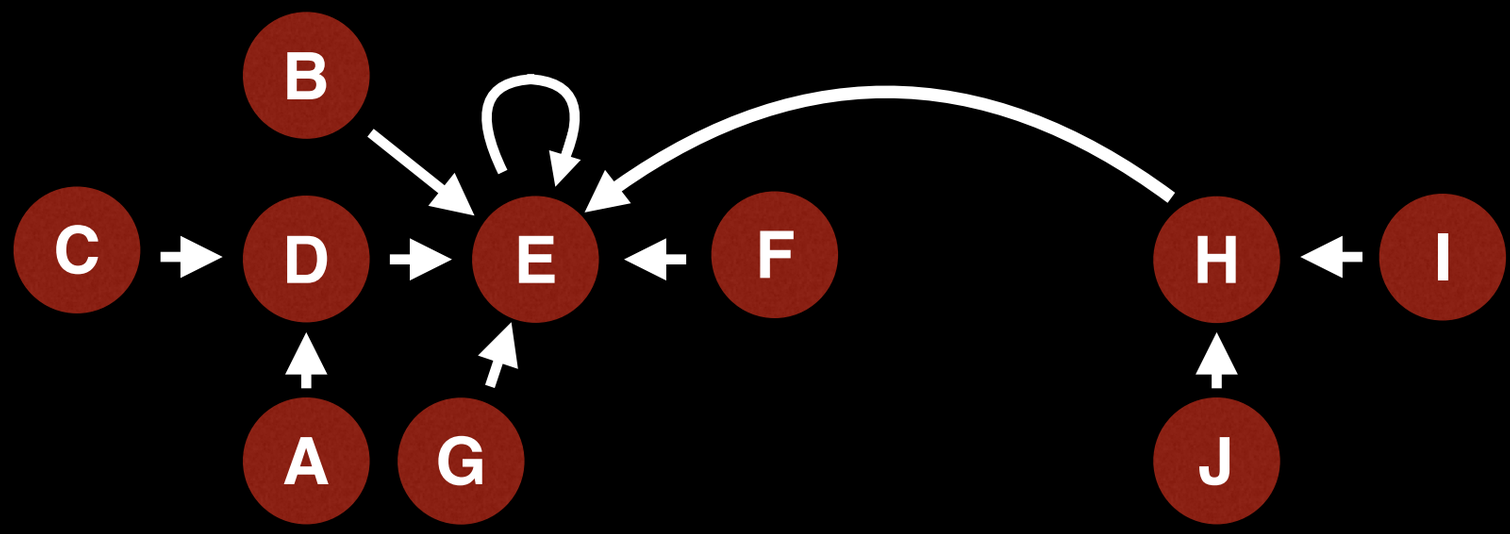


Using **path compression**

Instructions:

```
Union(A,B)
Union(C,D)
Union(E,F)
Union(G,H)
Union(I,J)
```

```
Union(J,G)
Union(H,F)
Union(A,C)
Union(D,E)
Union(G,B)
Union(I,J)
```

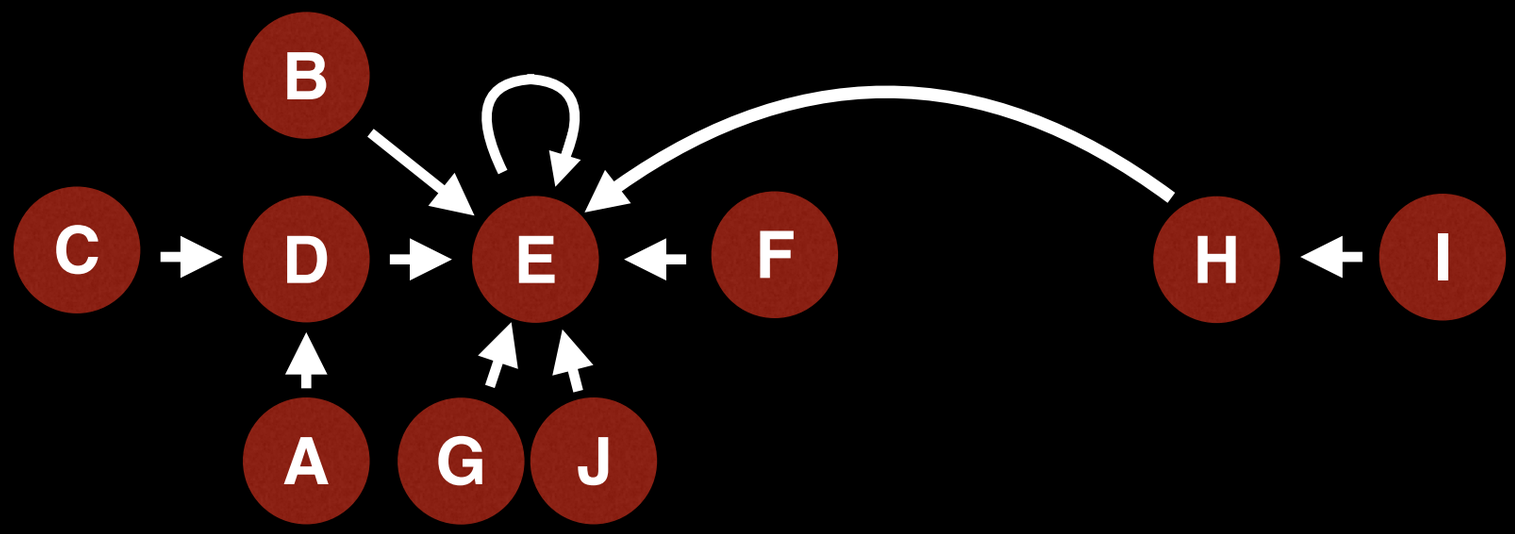


Using **path compression**

Instructions:

```
Union(A,B)
Union(C,D)
Union(E,F)
Union(G,H)
Union(I,J)
```

```
Union(J,G)
Union(H,F)
Union(A,C)
Union(D,E)
Union(G,B)
Union(I,J)
```

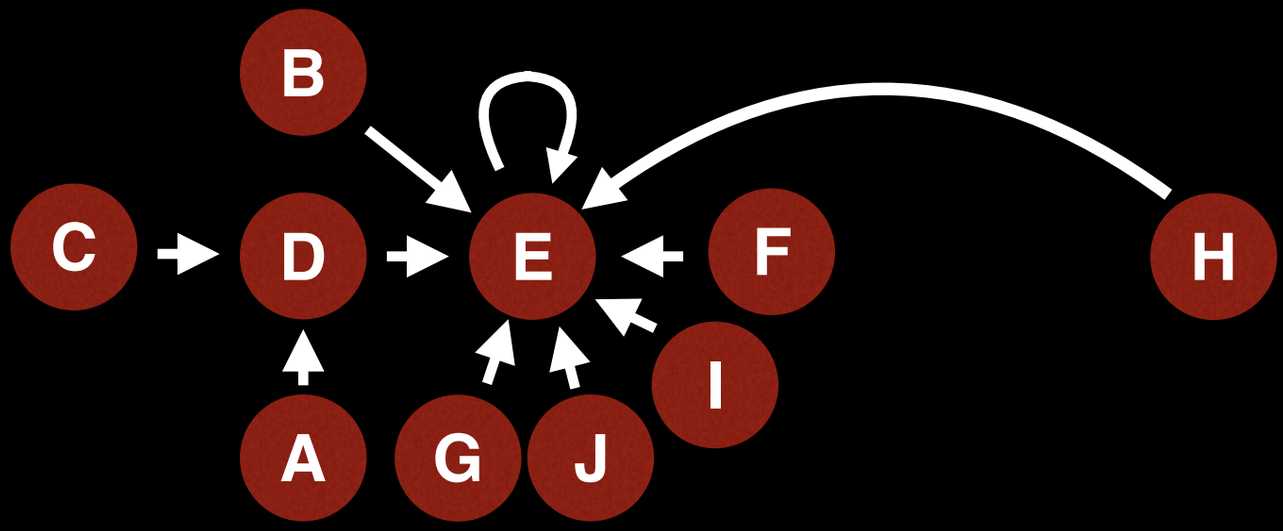


Using **path compression**

Instructions:

```
Union(A,B)
Union(C,D)
Union(E,F)
Union(G,H)
Union(I,J)
```

```
Union(J,G)
Union(H,F)
Union(A,C)
Union(D,E)
Union(G,B)
Union(I,J)
```



Instructions:

```
Union(A,B)
Union(C,D)
Union(E,F)
Union(G,H)
Union(I,J)
```

```
Union(J,G)
Union(H,F)
Union(A,C)
Union(D,E)
Union(G,B)
Union(I,J)
```

Union Find

Source Code

Fenwick Tree

(Binary Indexed Tree)

Discussion and Examples

Fenwick Tree Motivation

Given an array of integer values compute the range sum between index [i, j).

	0	1	2	3	4	5	6	7	8	9
A =	5	-3	6	1	0	-4	11	6	2	7

Fenwick Tree Motivation

Given an array of integer values compute the range sum between index [i, j).

	0	1	2	3	4	5	6	7	8	9
A =	5	-3	6	1	0	-4	11	6	2	7

Sum of A from [2,7) = 6 + 1 + 0 + −4 + 11 = 14

Fenwick Tree Motivation

Given an array of integer values compute the range sum between index [i, j).

	0	1	2	3	4	5	6	7	8	9
A =	5	-3	6	1	0	-4	11	6	2	7

Sum of A from [2,7) = 6 + 1 + 0 + −4 + 11 = 14

Sum of A from [0,4) = 5 + −3 + 6 + 1 = 9

Fenwick Tree Motivation

Given an array of integer values compute the range sum between index [i, j).

	0	1	2	3	4	5	6	7	8	9
A =	**5**	**-3**	**6**	**1**	**0**	**-4**	**11**	**6**	**2**	**7**

Sum of A from [2,7) = 6 + 1 + 0 + −4 + 11 = 14
Sum of A from [0,4) = 5 + −3 + 6 + 1 = 9
Sum of A from [7,8) = 6 = 6

Fenwick Tree Motivation

Given an array of integer values compute the range sum between index [i, j).

	0	1	2	3	4	5	6	7	8	9
A =	5	-3	6	1	0	-4	11	6	2	7

P =	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅

Let **P** be an array containing all the **prefix sums** of A.

Fenwick Tree Motivation

Given an array of integer values compute the range sum between index [i, j).

	0	1	2	3	4	5	6	7	8	9
A =	5	-3	6	1	0	-4	11	6	2	7

P =	0	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅

Let **P** be an array containing all the **prefix sums** of A.

Fenwick Tree Motivation

Given an array of integer values compute the range sum between index [i, j).

	0	1	2	3	4	5	6	7	8	9
A =	5	-3	6	1	0	-4	11	6	2	7

P =	0	5	∅	∅	∅	∅	∅	∅	∅	∅	∅

Let **P** be an array containing all the **prefix sums** of A.

Fenwick Tree Motivation

Given an array of integer values compute the range sum between index [i, j).

	0	1	2	3	4	5	6	7	8	9
A =	5	-3	6	1	0	-4	11	6	2	7

P =	0	5	2	∅	∅	∅	∅	∅	∅	∅	∅

Let **P** be an array containing all the **prefix sums** of A.

Fenwick Tree Motivation

Given an array of integer values compute the range sum between index [i, j).

	0	1	2	3	4	5	6	7	8	9
A =	5	-3	6	1	0	-4	11	6	2	7

P =	0	5	2	8	∅	∅	∅	∅	∅	∅	∅

Let **P** be an array containing all the **prefix sums** of A.

Fenwick Tree Motivation

Given an array of integer values compute the range sum between index [i, j).

	0	1	2	3	4	5	6	7	8	9
A =	5	-3	6	1	0	-4	11	6	2	7

P =	0	5	2	8	9	∅	∅	∅	∅	∅	∅

Let **P** be an array containing all the **prefix sums** of A.

Fenwick Tree Motivation

Given an array of integer values compute the range sum between index [i, j).

	0	1	2	3	4	5	6	7	8	9
A =	5	-3	6	1	0	-4	11	6	2	7

P =	0	5	2	8	9	9	∅	∅	∅	∅	∅

Let **P** be an array containing all the **prefix sums** of A.

Fenwick Tree Motivation

Given an array of integer values compute the range sum between index [i, j).

	0	1	2	3	4	5	6	7	8	9
A =	5	-3	6	1	0	-4	11	6	2	7

P =	0	5	2	8	9	9	5	∅	∅	∅	∅

Let **P** be an array containing all the **prefix sums** of A.

Fenwick Tree Motivation

Given an array of integer values compute the range sum between index [i, j).

	0	1	2	3	4	5	6	7	8	9
A =	5	-3	6	1	0	-4	11	6	2	7

P =	0	5	2	8	9	9	5	16	∅	∅	∅

Let **P** be an array containing all the **prefix sums** of A.

Fenwick Tree Motivation

Given an array of integer values compute the range sum between index [i, j).

	0	1	2	3	4	5	6	7	8	9
A =	5	-3	6	1	0	-4	11	6	2	7

P =	0	5	2	8	9	9	5	16	22	∅	∅

Let **P** be an array containing all the **prefix sums** of A.

Fenwick Tree Motivation

Given an array of integer values compute the range sum between index [i, j).

	0	1	2	3	4	5	6	7	8	9
A =	5	-3	6	1	0	-4	11	6	2	7

P =	0	5	2	8	9	9	5	16	22	24	∅

Let **P** be an array containing all the **prefix sums** of A.

Fenwick Tree Motivation

Given an array of integer values compute the range sum between index [i, j).

	0	1	2	3	4	5	6	7	8	9
A =	5	-3	6	1	0	-4	11	6	2	7

P =	0	5	2	8	9	9	5	16	22	24	31

Let **P** be an array containing all the **prefix sums** of A.

Fenwick Tree Motivation

Given an array of integer values compute the range sum between index [i, j).

	0	1	2	3	4	5	6	7	8	9
A =	5	-3	6	1	0	-4	11	6	2	7

P =	0	5	2	8	9	9	5	16	22	24	31

Sum of A from [2,7) = P[7] – P[2] = 16 – 2 = 14

Fenwick Tree Motivation

Given an array of integer values compute the range sum between index [i, j).

	0	1	2	3	4	5	6	7	8	9
A =	5	-3	6	1	0	-4	11	6	2	7

P =	0	5	2	8	9	9	5	16	22	24	31

Sum of A from [2,7) = P[7] – P[2] = 16 – 2 = 14

Sum of A from [0,4) = P[4] – P[0] = 9 – 0 = 9

Fenwick Tree Motivation

Given an array of integer values compute the range sum between index [i, j).

	0	1	2	3	4	5	6	7	8	9
A =	5	-3	6	1	0	-4	11	6	2	7

P =	0	5	2	8	9	9	5	16	22	24	31

Sum of A from [2,7) = P[7] – P[2] = 16 – 2 = 14

Sum of A from [0,4) = P[4] – P[0] = 9 – 0 = 9

Sum of A from [7,8) = P[8] – P[7] = 22 – 16 = 6

Fenwick Tree Motivation

Question: What about if we want to update our initial array with some new value?

	0	1	2	3	4	5	6	7	8	9
A =	5	-3	6	1	0	-4	11	6	2	7

P =	0	5	2	8	9	9	5	16	22	24	31

Fenwick Tree Motivation

Question: What about if we want to update our initial array with some new value?

	0	1	2	3	4	5	6	7	8	9
A =	5	-3	6	1	3	-4	11	6	2	7

P =	0	5	2	8	9	12	8	19	25	27	34

A[4] = 3

Fenwick Tree Motivation

Question: What about if we want to update our initial array with some new value?

	0	1	2	3	4	5	6	7	8	9
A =	5	-3	6	1	3	-4	11	6	2	7

	0	1	2	3	4	5	6	7	8	9	10
P =	0	5	2	8	9	12	8	19	25	27	34

A prefix sum array is great for **static arrays**, but takes **O(n)** for updates.

What is a Fenwick Tree?

A **Fenwick Tree** (also called Binary Indexed Tree) is a data structure that supports sum range queries as well as setting values in a static array and getting the value of the prefix sum up some index efficiently.

Complexity

Construction	O(n)
Point Update	O(log(n))
Range Sum	O(log(n))
Range Update	O(log(n))
Adding Index	N/A
Removing Index	N/A

Fenwick Tree Range Queries

16 10000_2

15 01111_2

14 01110_2

13 01101_2

12 01100_2

11 01011_2

10 01010_2

9 01001_2

8 01000_2

7 00111_2

6 00110_2

5 00101_2

4 00100_2

3 00011_2

2 00010_2

1 00001_2

Unlike a regular array, in a Fenwick tree a specific cell is responsible for other cells as well.

16	10000_2
15	01111_2
14	01110_2
13	01101_2
12	01100_2
11	01011_2
10	01010_2
9	01001_2
8	01000_2
7	00111_2
6	00110_2
5	00101_2
4	00100_2
3	00011_2
2	00010_2
1	00001_2

Unlike a regular array, in a Fenwick tree a specific cell is responsible for other cells as well.

The position of the **least significant bit** (**LSB**) determines the range of responsibility that cell has to the cells below itself.

16	10000_2
15	01111_2
14	01110_2
13	01101_2
12	01100_2
11	01011_2
10	01010_2
9	01001_2
8	01000_2
7	00111_2
6	00110_2
5	00101_2
4	00100_2
3	00011_2
2	00010_2
1	00001_2

Unlike a regular array, in a Fenwick tree a specific cell is responsible for other cells as well.

The position of the **least significant bit** (**LSB**) determines the range of responsibility that cell has to the cells below itself.

Index 12 in binary is: 1100_2

LSB is at position 3, so this index is responsible for $2^{3-1} = 4$ cells below itself.

16	10000_2
15	01111_2
14	01110_2
13	01101_2
12	01100_2
11	01011_2
10	01010_2
9	01001_2
8	01000_2
7	00111_2
6	00110_2
5	00101_2
4	00100_2
3	00011_2
2	00010_2
1	00001_2

Unlike a regular array, in a Fenwick tree a specific cell is responsible for other cells as well.

The position of the **least significant bit** (**LSB**) determines the range of responsibility that cell has to the cells below itself.

Index 10 in binary is: 1010_2

LSB is at position 2, so this index is responsible for $2^{2-1} = 2$ cells below itself.

16	10000_2
15	01111_2
14	01110_2
13	01101_2
12	01100_2
11	01011_2
10	01010_2
9	01001_2
8	01000_2
7	00111_2
6	00110_2
5	00101_2
4	00100_2
3	00011_2
2	00010_2
1	00001_2

Unlike a regular array, in a Fenwick tree a specific cell is responsible for other cells as well.

The position of the **least significant bit** (**LSB**) determines the range of responsibility that cell has to the cells below itself.

Index 11 in binary is: 1011_2

LSB is at position 1, so this index is responsible for $2^{1-1} = 1$ cell (itself).

16 10000_2

15 01111_2 |

14 01110_2

13 01101_2 |

12 01100_2

11 01011_2 |

10 01010_2

9 01001_2 |

8 01000_2

7 00111_2 |

6 00110_2

5 00101_2 |

4 00100_2

3 00011_2 |

2 00010_2

1 00001_2 |

Blue bars | represent the **range of responsibility** for that cell **NOT value**.

All odd numbers have a their first least significant bit set in the ones position, so they are only responsible for themselves.

16 10000_2

15 01111_2

14 01110_2

13 01101_2

12 01100_2

11 01011_2

10 01010_2

9 01001_2

8 01000_2

7 00111_2

6 00110_2

5 00101_2

4 00100_2

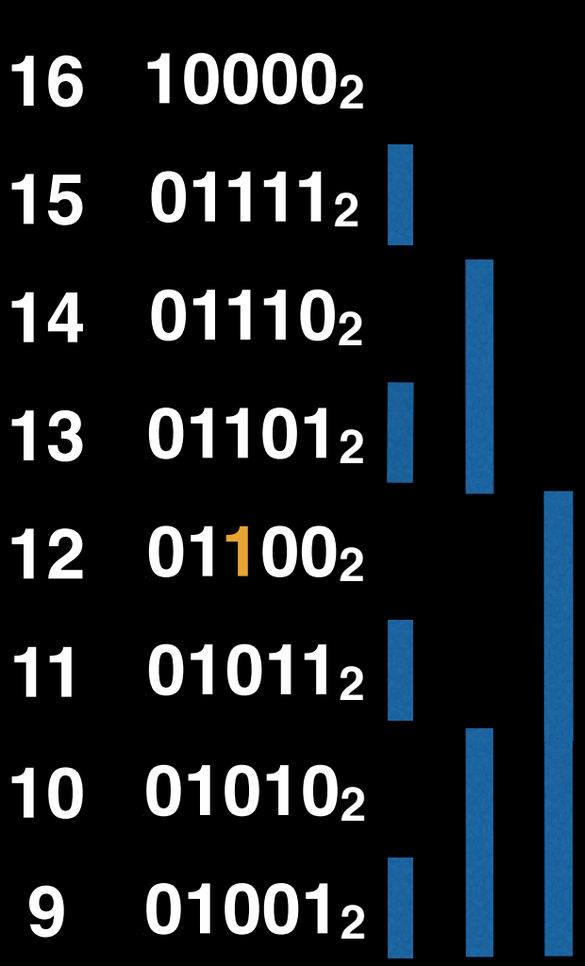
3 00011_2

2 00010_2

1 00001_2

Blue bars represent the **range of responsibility** for that cell **NOT value**.

Numbers with their least significant bit in the second position have a range of two.



Blue bars | represent the **range of responsibility** for that cell **NOT value**.

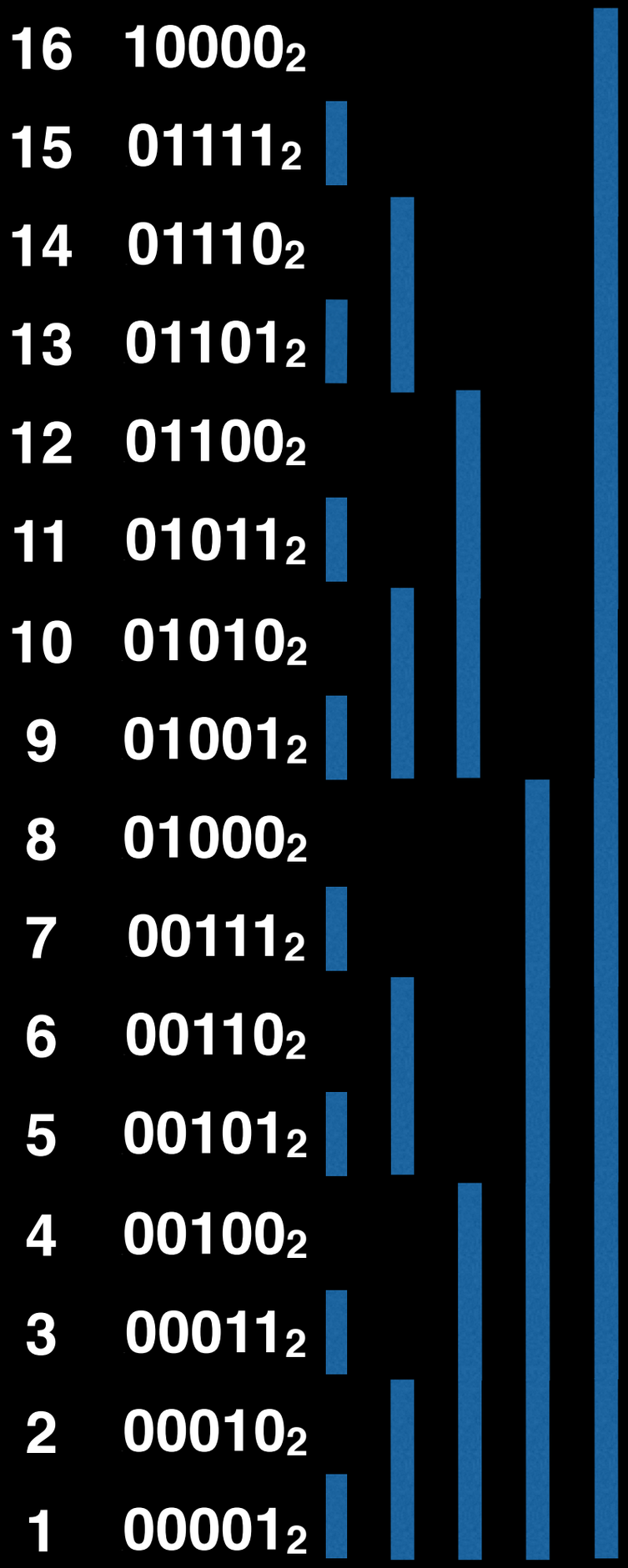
Numbers with their least significant bit in the third position have a range of four.

Blue bars represent the **range of responsibility** for that cell **NOT value**.

Numbers with their least significant bit in the fourth position have a range of eight.

Blue bars | represent the **range of responsibility** for that cell **NOT value**.

Numbers with their least significant bit in the fifth position have a range of sixteen.



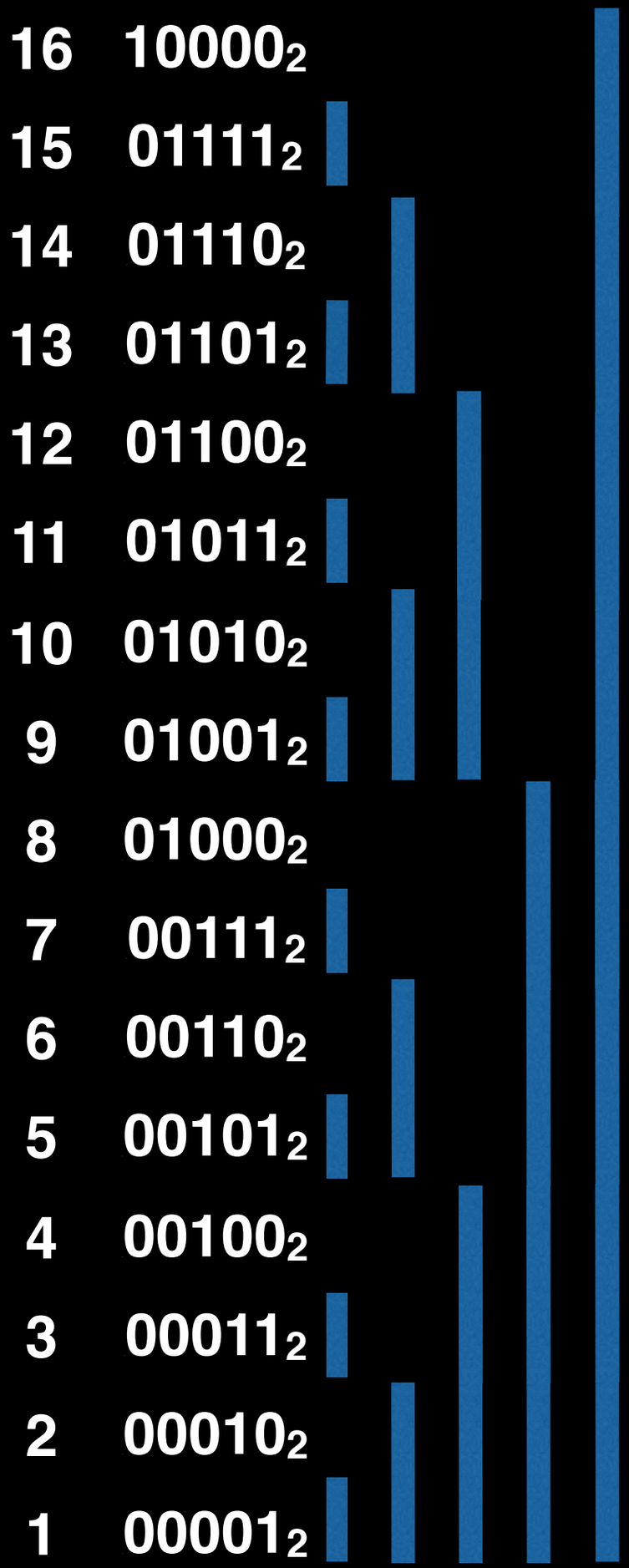
Range Queries

In a Fenwick tree we may compute the **prefix sum** up to a certain index, which ultimately lets us perform range sum queries.

Range Queries

In a Fenwick tree we may compute the **prefix sum** up to a certain index, which ultimately lets us perform range sum queries.

Idea: Suppose you want to find the prefix sum of [1, i], then you **start at i and cascade downwards** until you reach zero adding the value at each of the indices you encounter.



Range Queries

In a Fenwick tree we may compute the **prefix sum** up to a certain index, which ultimately lets us perform range sum queries.

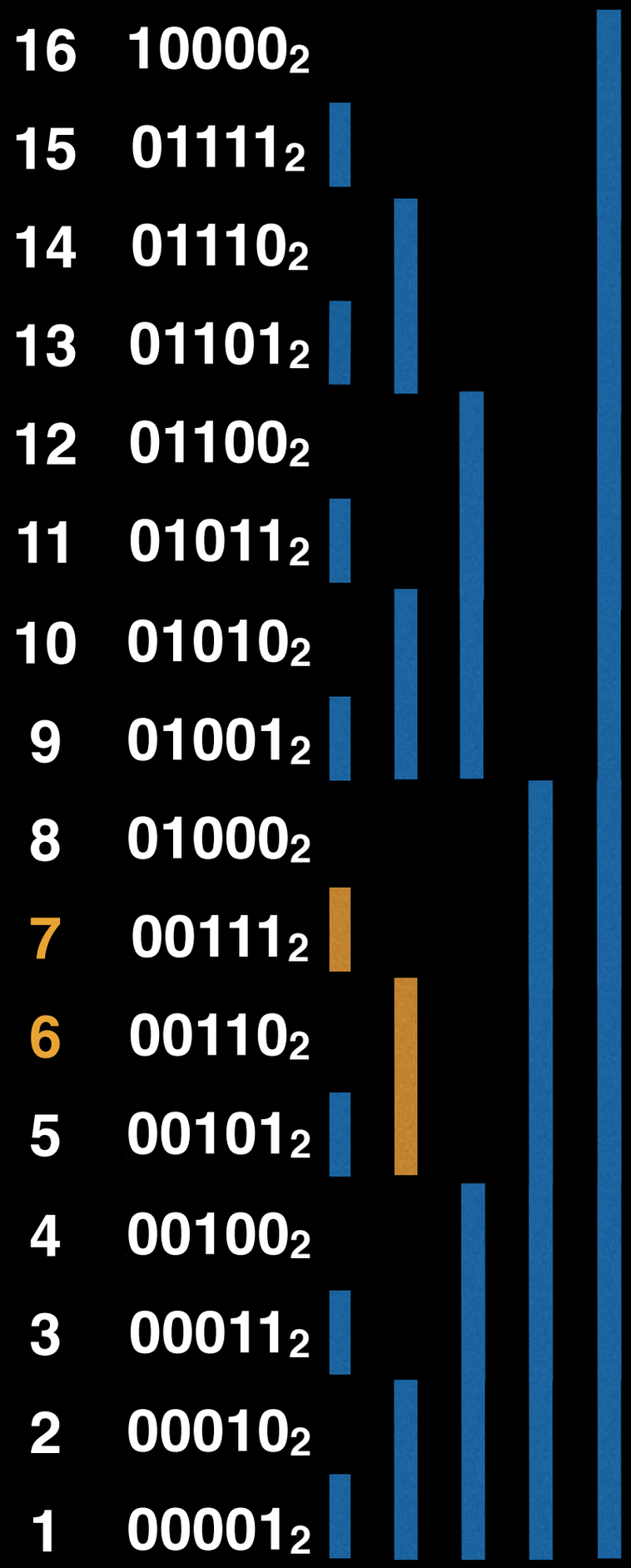
Find the prefix sum up to index 7.

Range Queries

In a Fenwick tree we may compute the **prefix sum** up to a certain index, which ultimately lets us perform range sum queries.

Find the prefix sum up to index 7.

```
sum = A[7]
```

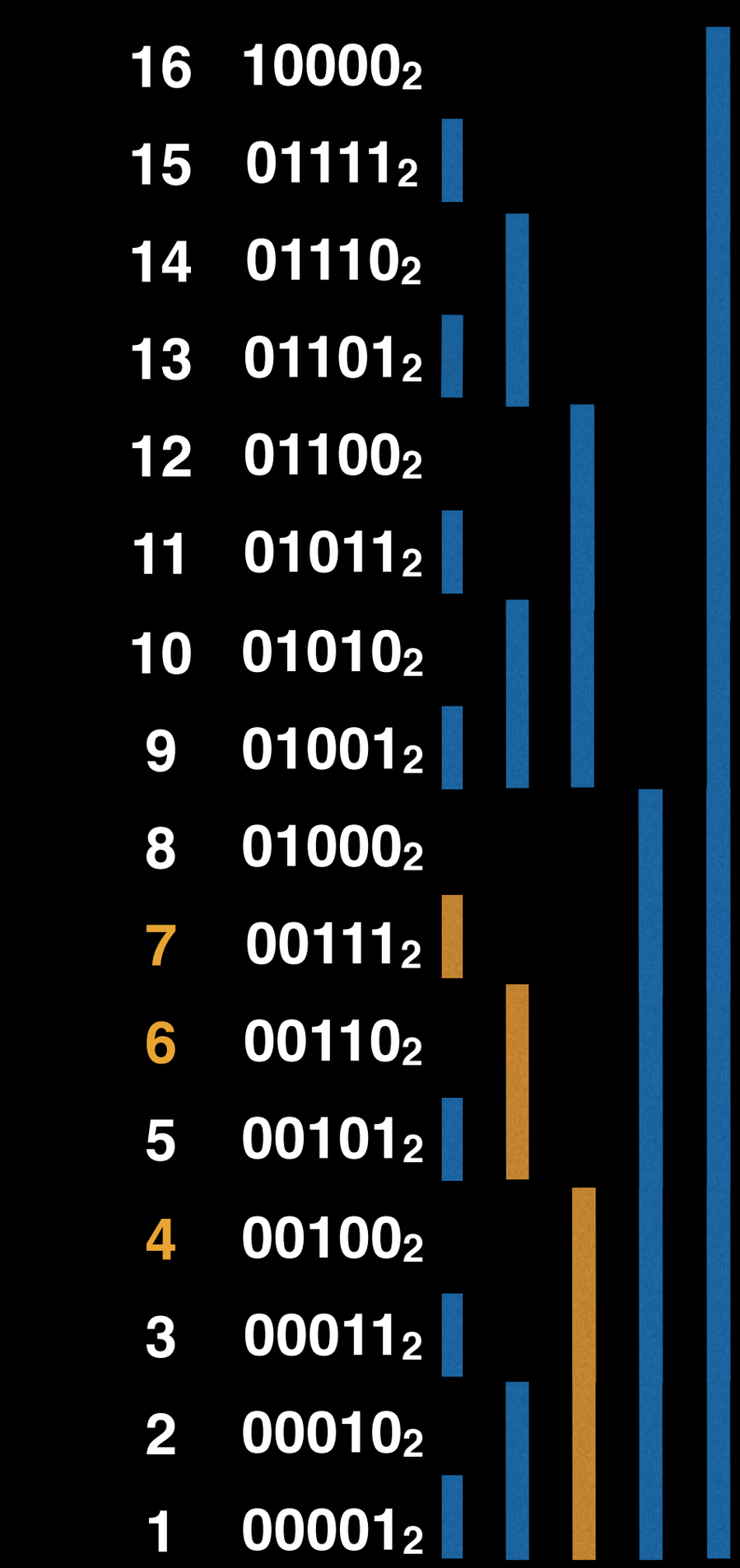


Range Queries

In a Fenwick tree we may compute the **prefix sum** up to a certain index, which ultimately lets us perform range sum queries.

Find the prefix sum up to index 7.

```
sum = A[7] + A[6]
```

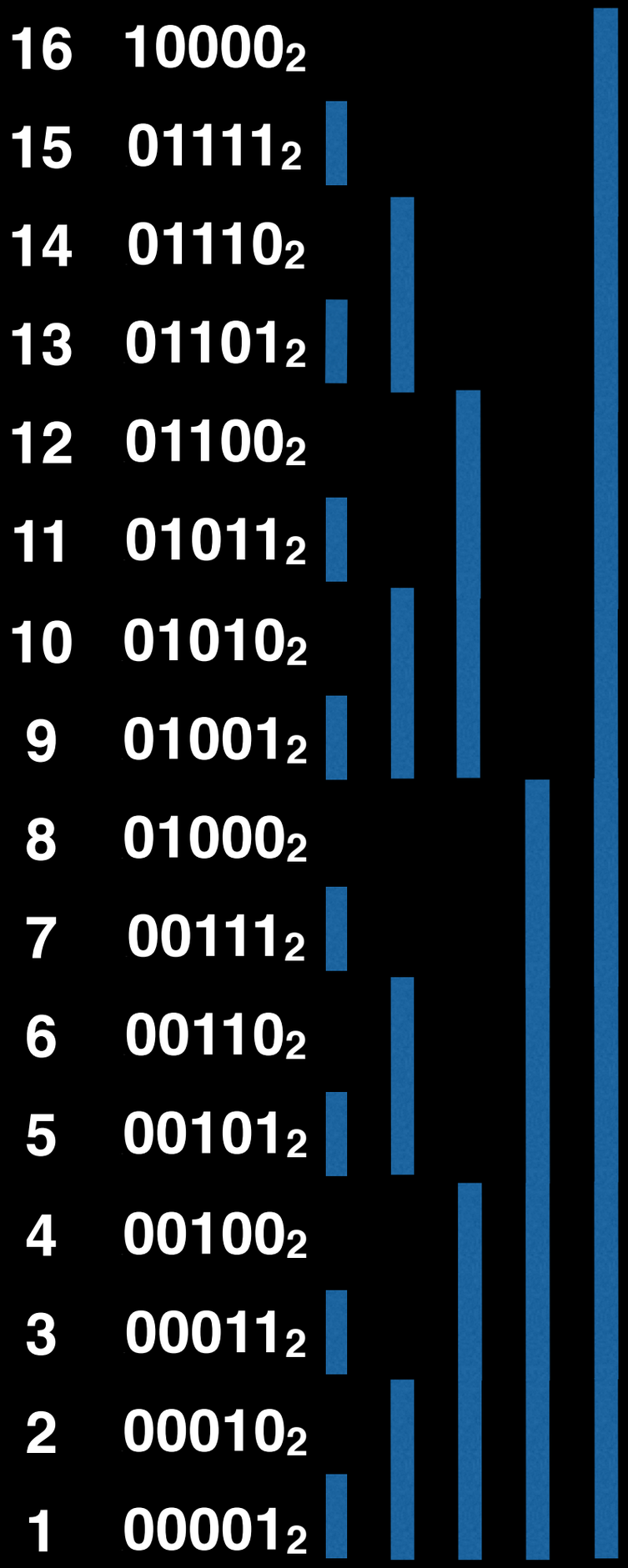


Range Queries

In a Fenwick tree we may compute the **prefix sum** up to a certain index, which ultimately lets us perform range sum queries.

Find the prefix sum up to index 7.

```
sum = A[7] + A[6] + A[4]
```



Range Queries

In a Fenwick tree we may compute the **prefix sum** up to a certain index, which ultimately lets us perform range sum queries.

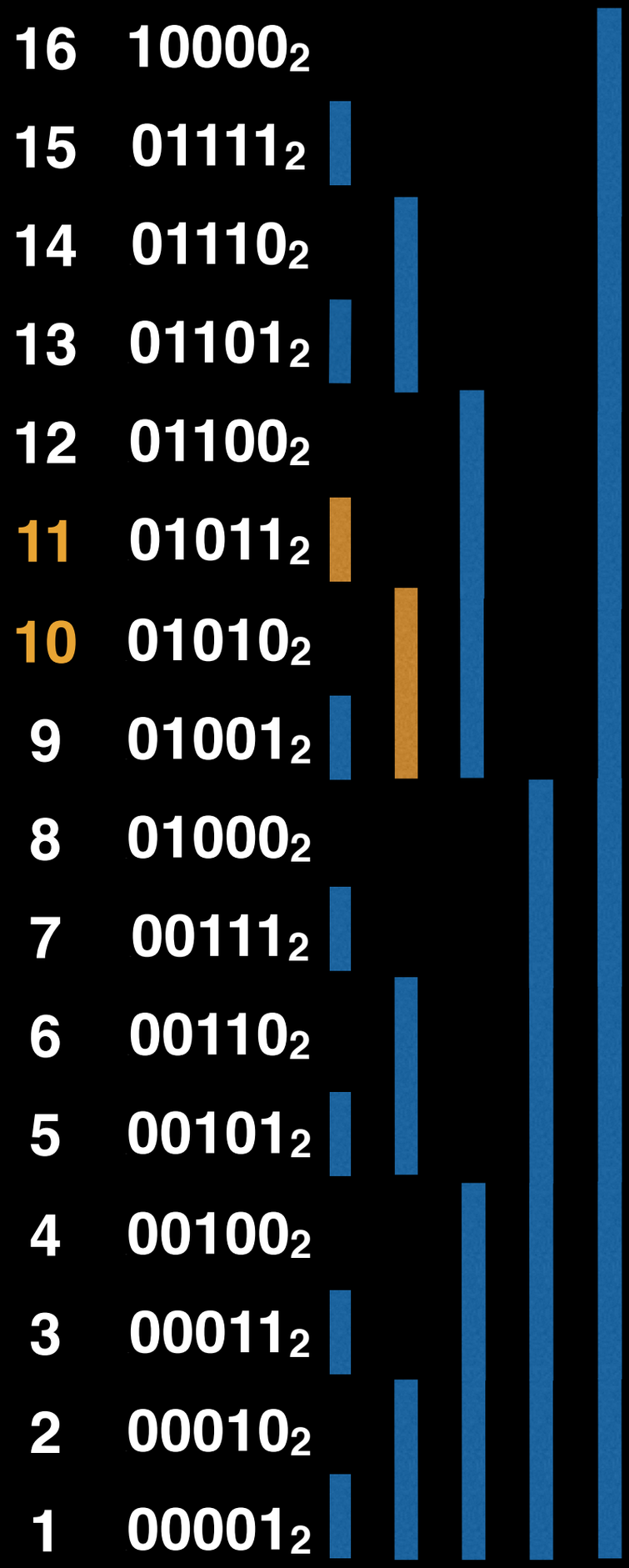
Find the prefix sum up to index 11.

Range Queries

In a Fenwick tree we may compute the **prefix sum** up to a certain index, which ultimately lets us perform range sum queries.

Find the prefix sum up to index 11.

```
sum = A[11]
```

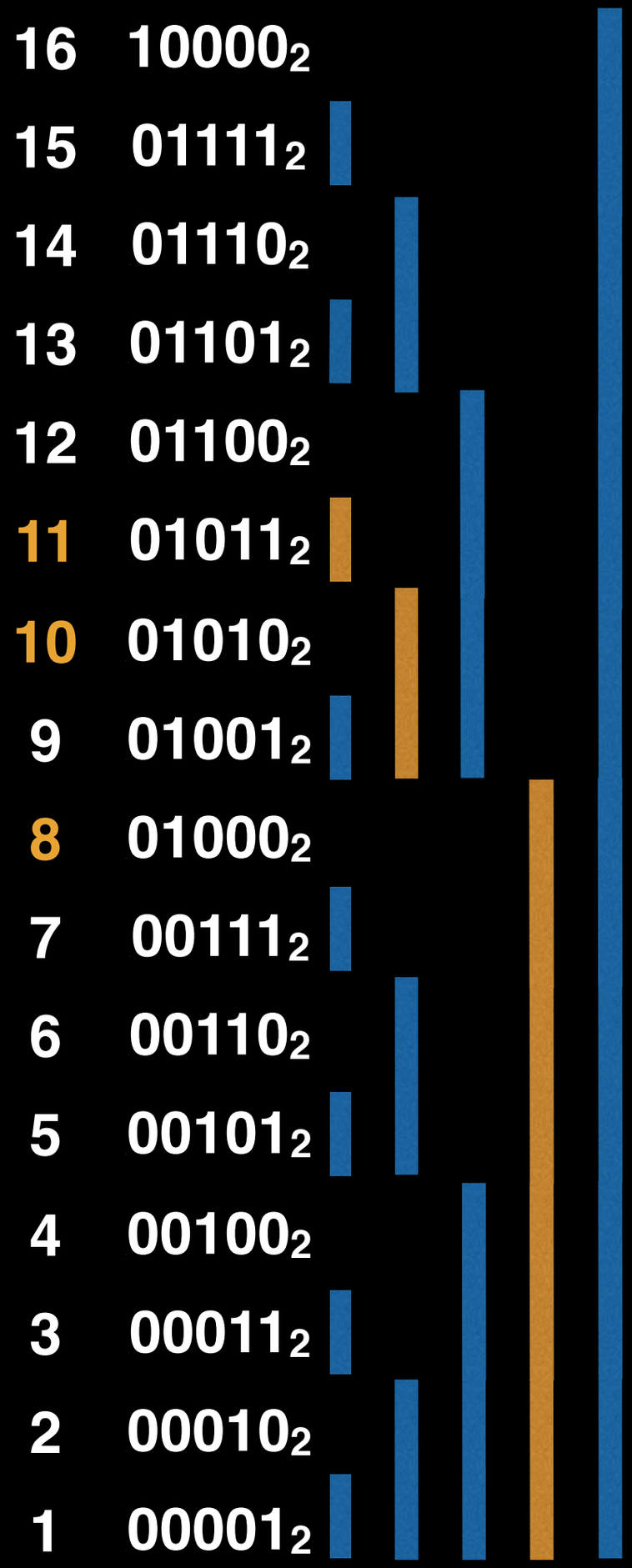


Range Queries

In a Fenwick tree we may compute the **prefix sum** up to a certain index, which ultimately lets us perform range sum queries.

Find the prefix sum up to index 11.

```
sum = A[11] + A[10]
```

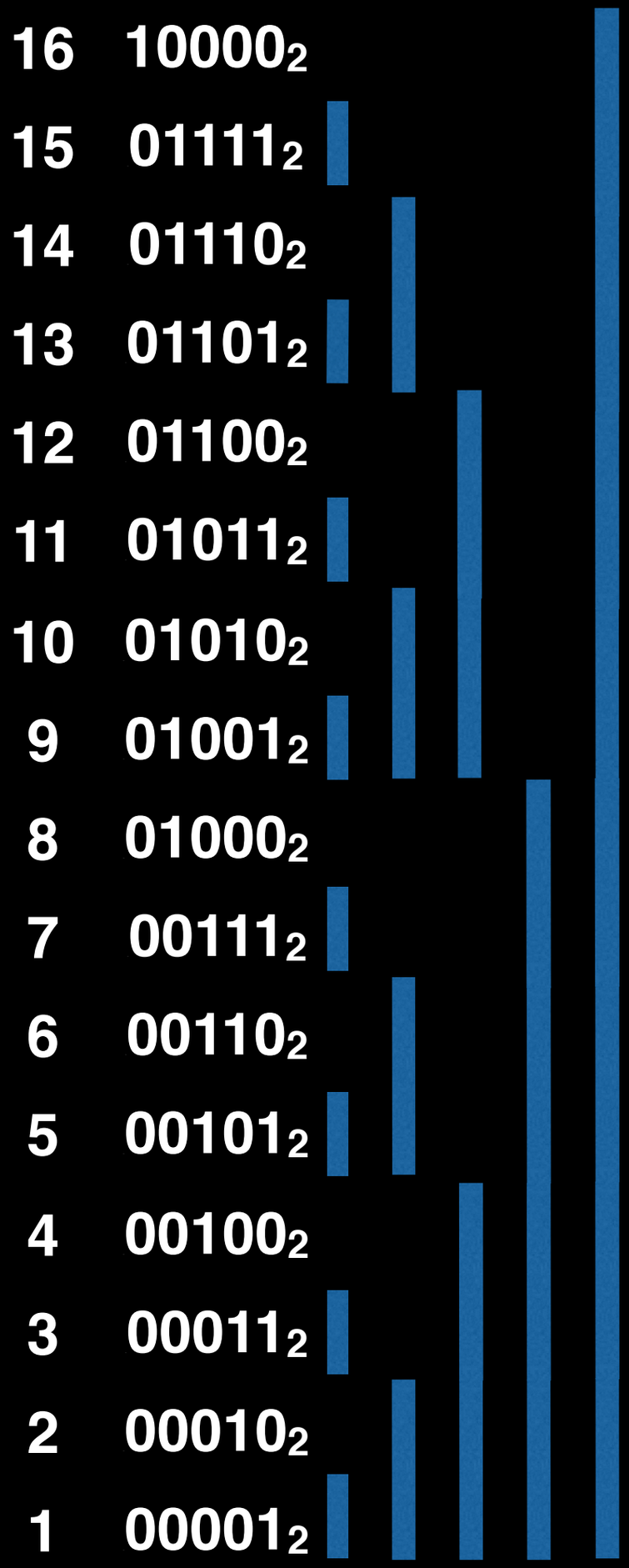


Range Queries

In a Fenwick tree we may compute the **prefix sum** up to a certain index, which ultimately lets us perform range sum queries.

Find the prefix sum up to index 11.

```
sum = A[11] + A[10] + A[8]
```



Range Queries

In a Fenwick tree we may compute the **prefix sum** up to a certain index, which ultimately lets us perform range sum queries.

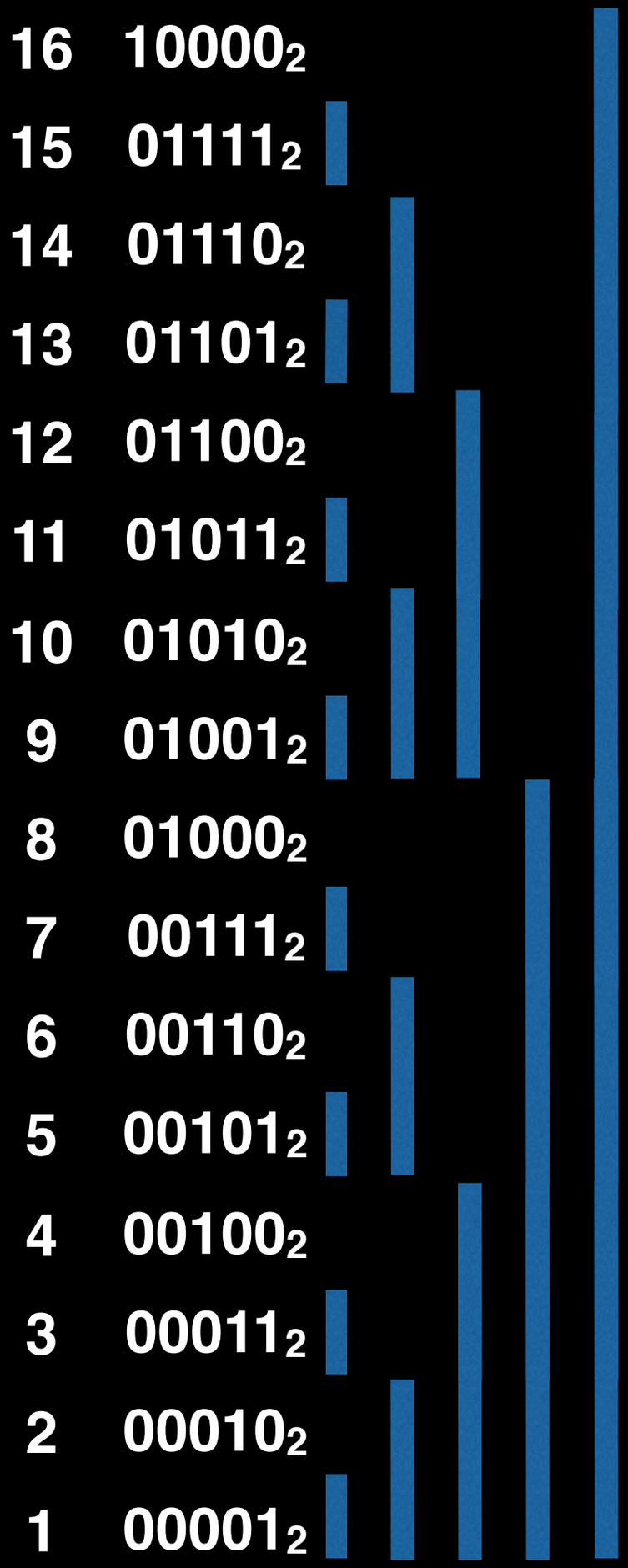
Find the prefix sum up to index 4.

Range Queries

In a Fenwick tree we may compute the **prefix sum** up to a certain index, which ultimately lets us perform range sum queries.

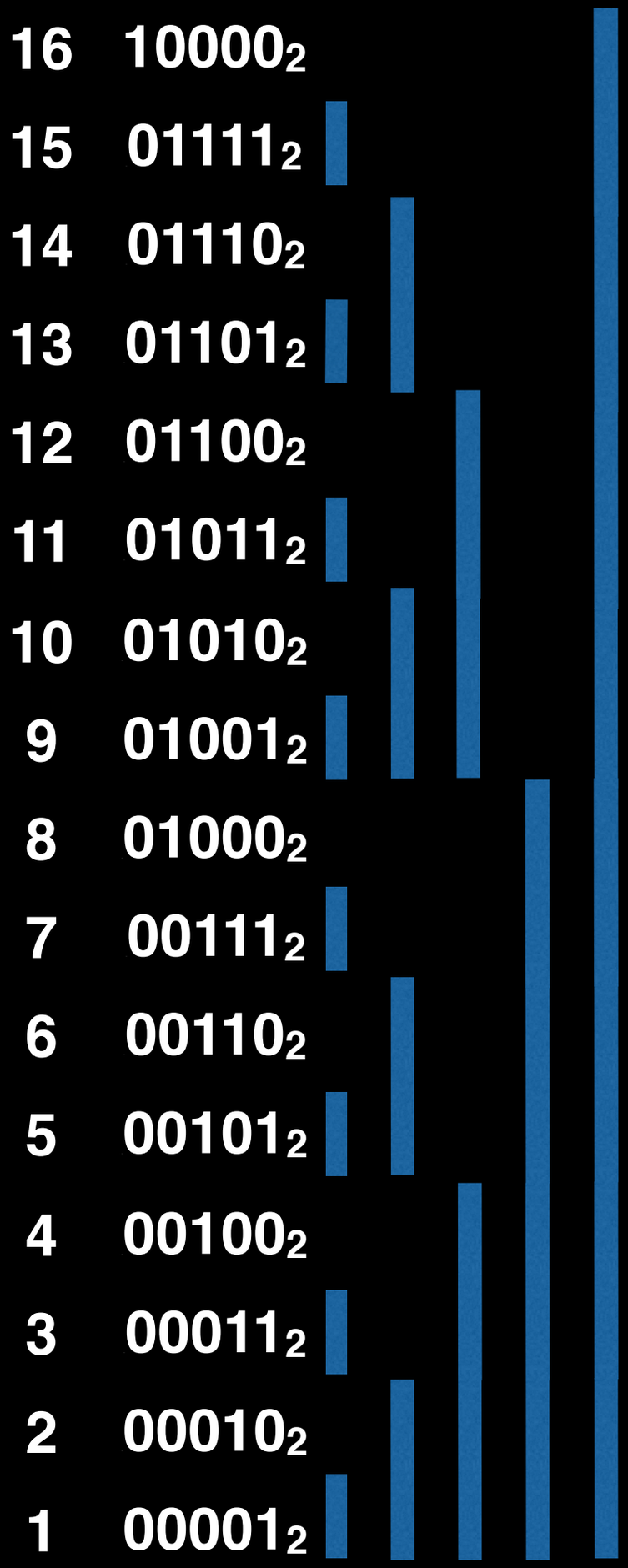
Find the prefix sum up to index 4.

sum = A[4]



Range Queries

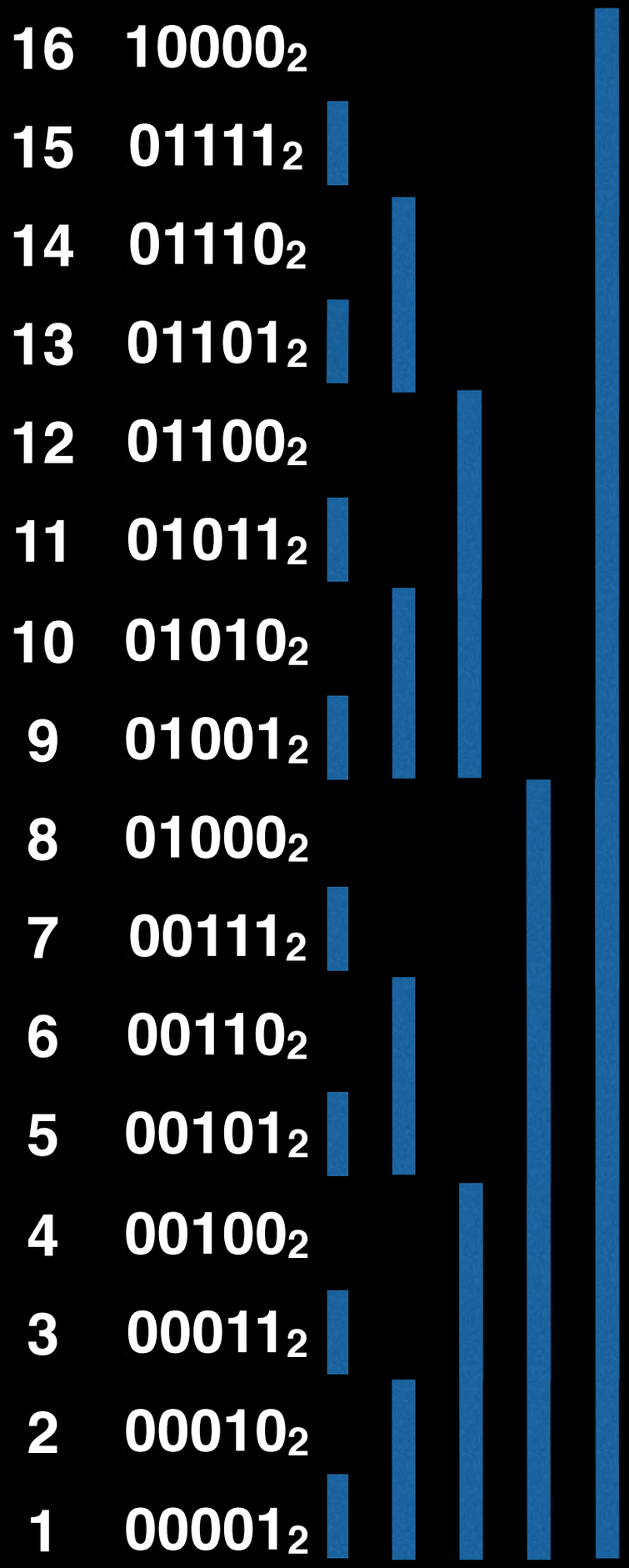
In a Fenwick tree we may compute the **prefix sum** up to a certain index, which ultimately lets us perform range sum queries.



Range Queries

Let’s use prefix sums to compute the interval sum between [i, j].

Compute the interval sum between [11, 15].



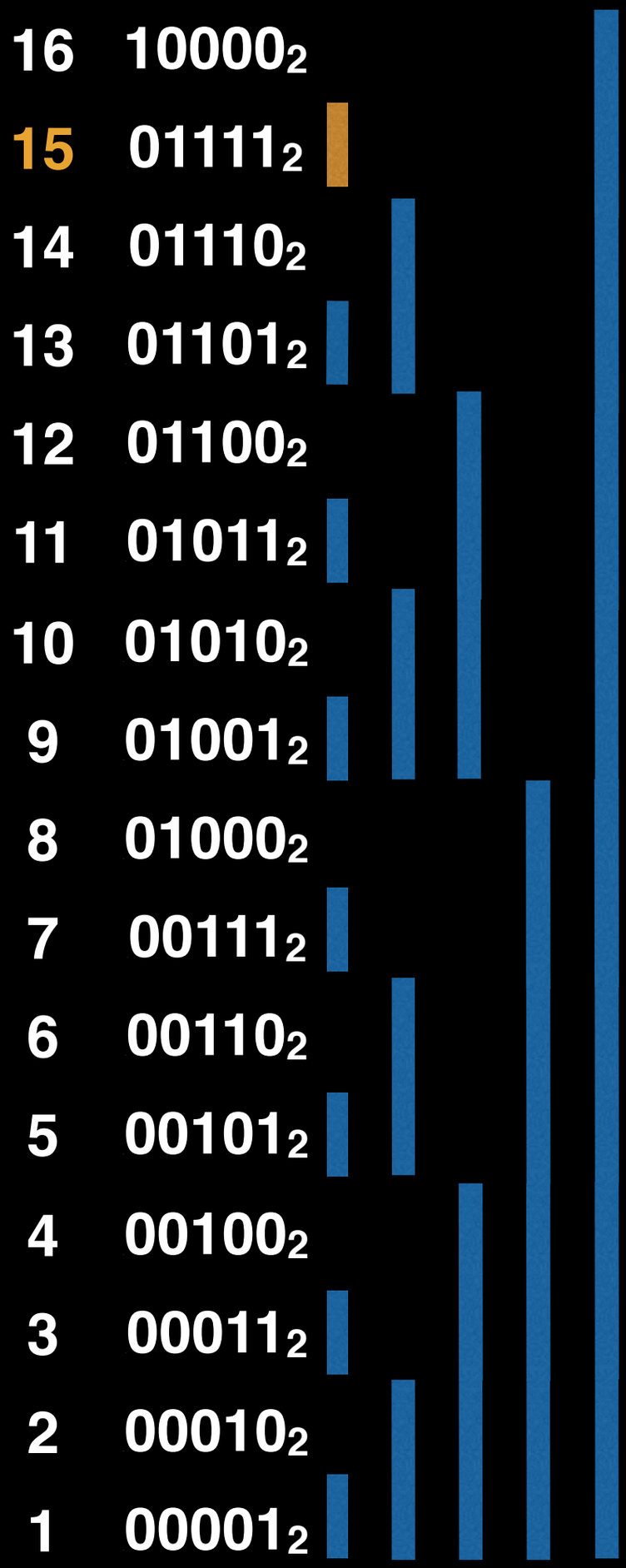
Range Queries

Let’s use prefix sums to compute the interval sum between [i, j].

Compute the interval sum between [11, 15].

First we compute the prefix sum of [1, 15], then we will compute the prefix sum of [1,11) and get the difference.

Not inclusive! We want the value at position 11.



Range Queries

Compute the interval sum between [11, 15].

First we compute the prefix sum of [1, 15], then we will compute the prefix sum of [1,11) and get the difference.

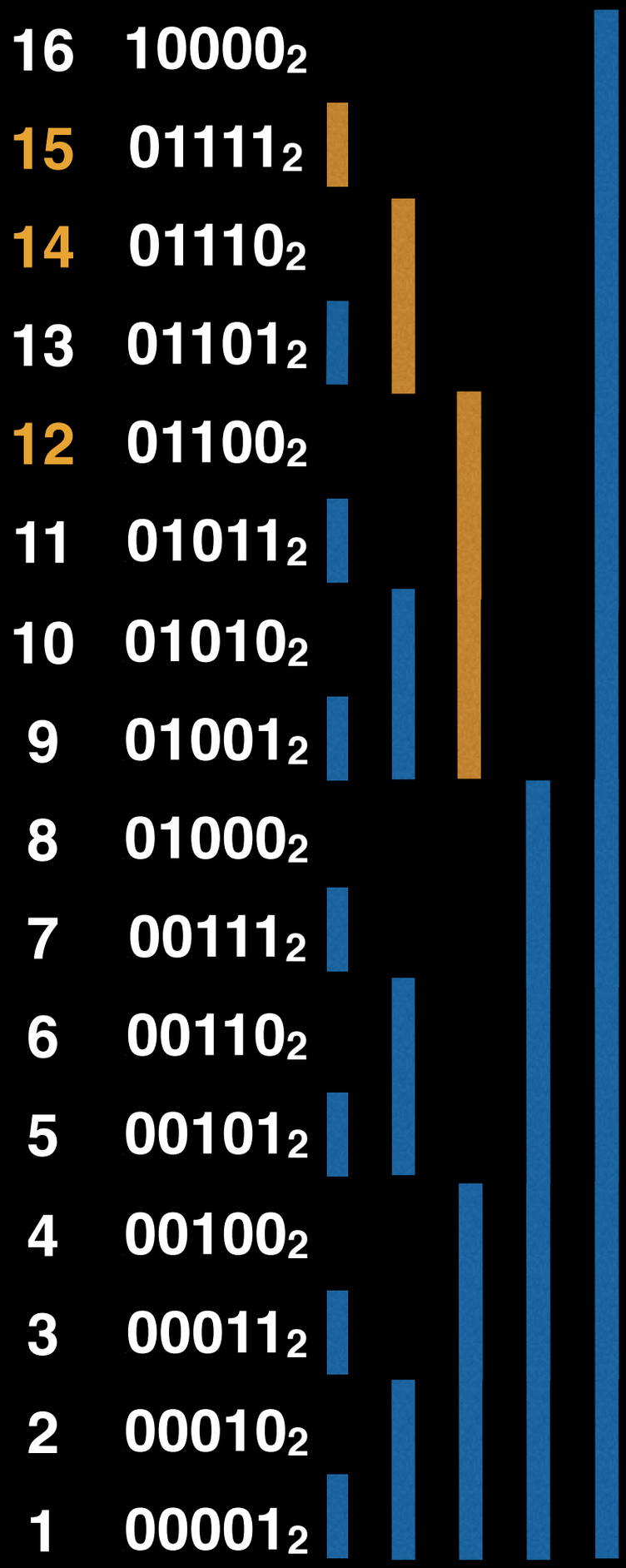
Sum of [1,15] = A[15]

Range Queries

Compute the interval sum between [11, 15].

First we compute the prefix sum of [1, 15], then we will compute the prefix sum of [1,11) and get the difference.

Sum of [1,15] = A[15]+A[14]



Range Queries

Compute the interval sum between [11, 15].

First we compute the prefix sum of [1, 15], then we will compute the prefix sum of [1,11) and get the difference.

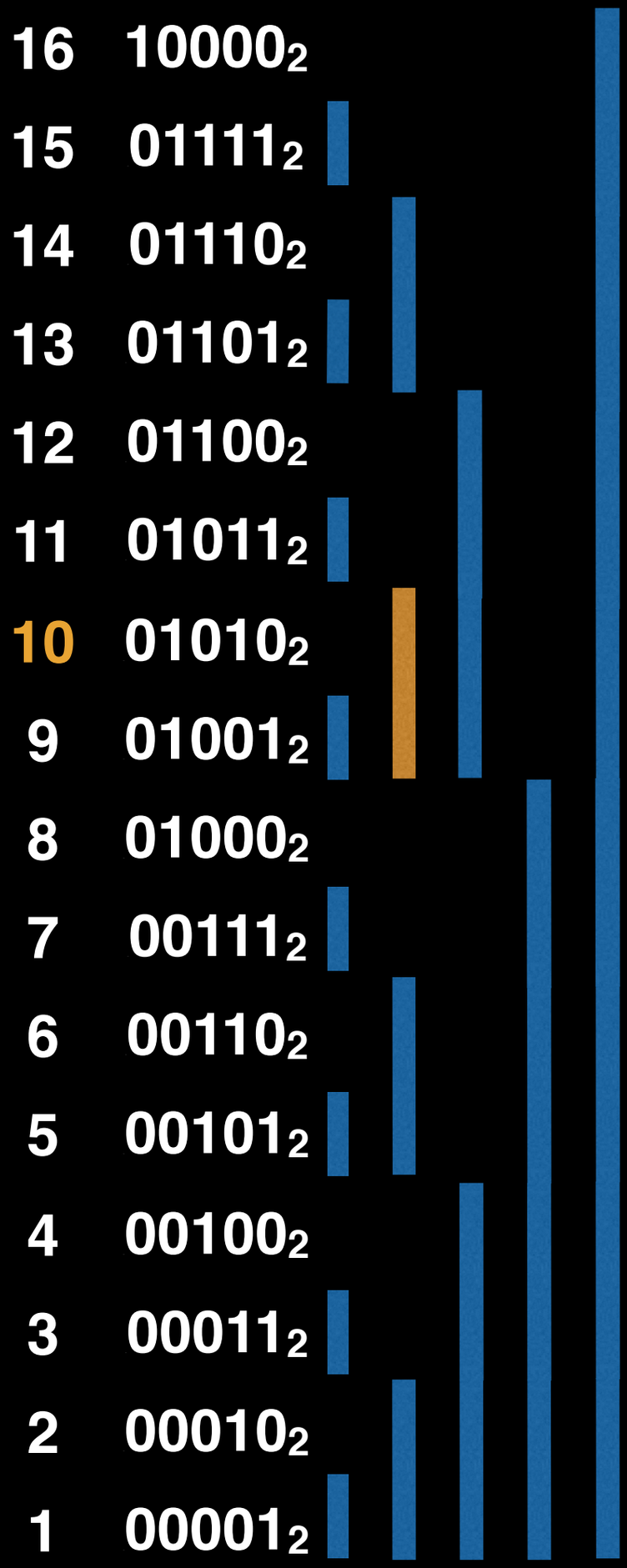
Sum of [1,15] = A[15]+A[14]+A[12]

Range Queries

Compute the interval sum between [11, 15].

First we compute the prefix sum of [1, 15], then we will compute the prefix sum of [1,11) and get the difference.

Sum of [1,15] = A[15]+A[14]+A[12]+A[8]



Range Queries

Compute the interval sum between [11, 15].

First we compute the prefix sum of [1, 15], then we will compute the prefix sum of [1,11) and get the difference.

Sum of [1,15] = A[15]+A[14]+A[12]+A[8]

Sum of [1,11) = A[10]

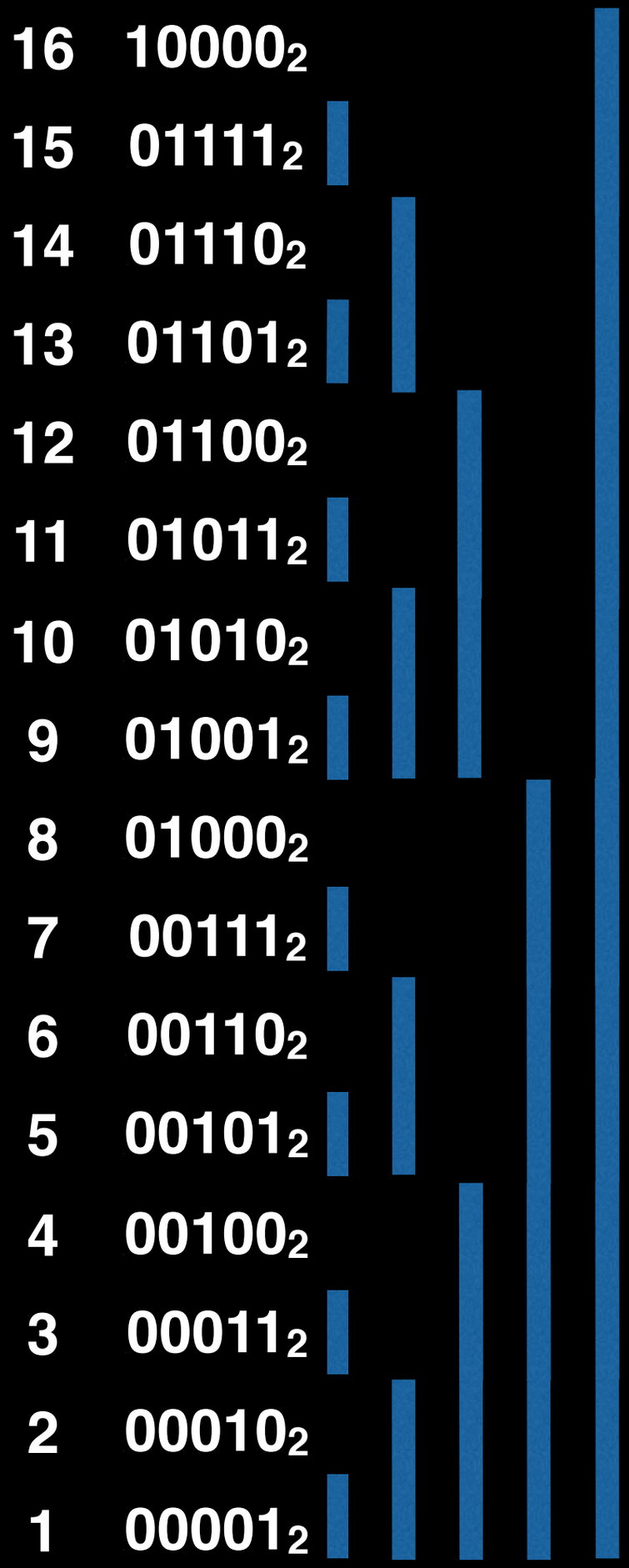
Range Queries

Compute the interval sum between [11, 15].

First we compute the prefix sum of [1, 15], then we will compute the prefix sum of [1,11) and get the difference.

Sum of [1,15] = A[15]+A[14]+A[12]+A[8]

Sum of [1,11) = A[10]+A[8]



Range Queries

Compute the interval sum between [11, 15].

First we compute the prefix sum of [1, 15], then we will compute the prefix sum of [1,11) and get the difference.

Sum of [1,15] = A[15]+A[14]+A[12]+A[8]

Sum of [1,11) = A[10]+A[8]

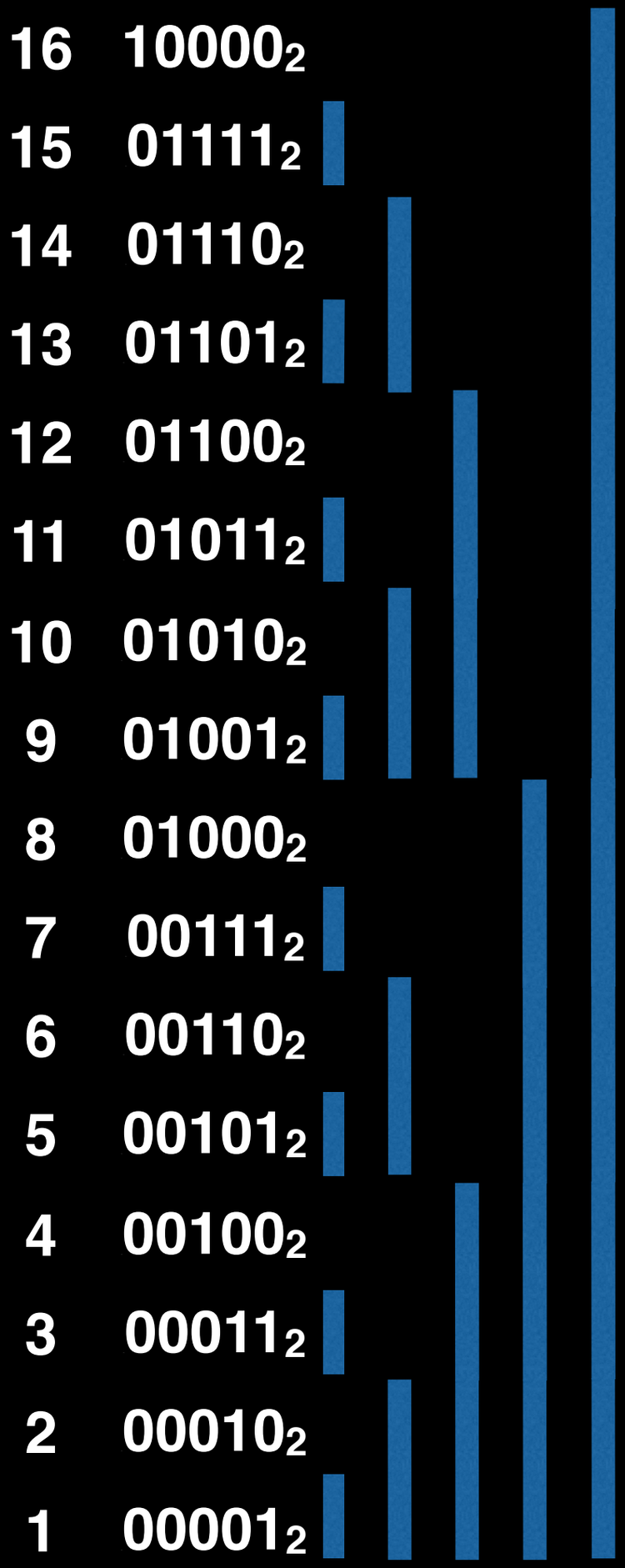
Range sum:

(A[15]+A[14]+A[12]+A[8])−(A[10]+A[8])

Fenwick Tree
Point Updates

Point Updates

Instead of querying a range to find the interval sum we want to update a cell in our array.



Point Updates

Instead of querying a range to find the interval sum we want to update a cell in our array.

Recall that with range queries we cascaded down from the current index by **continuously removing the LSB.**

Point Updates

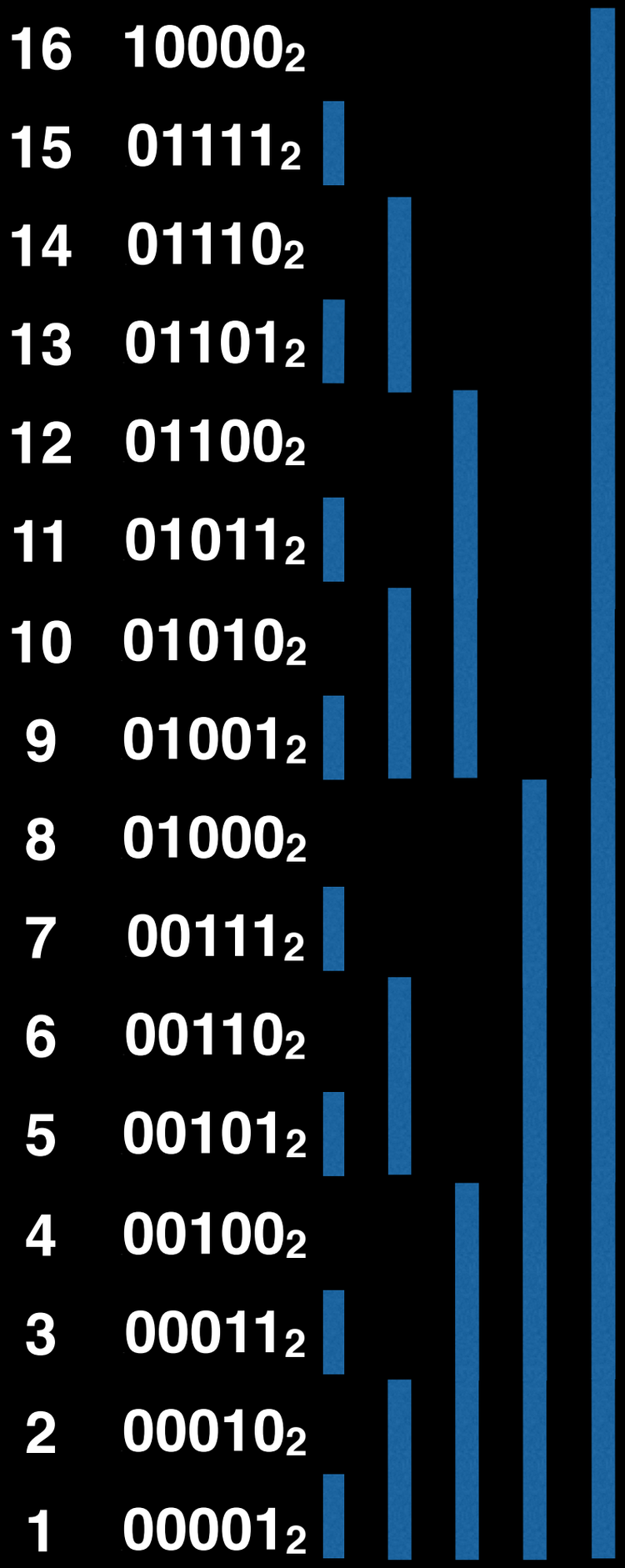
Instead of querying a range to find the interval sum we want to update a cell in our array.

Recall that with range queries we cascaded down from the current index by **continuously removing the LSB.**

$13 = 1101_2$, $1101_2 - 0001_2 = 1100_2$

$\downarrow$

$12 = 1100_2$



Point Updates

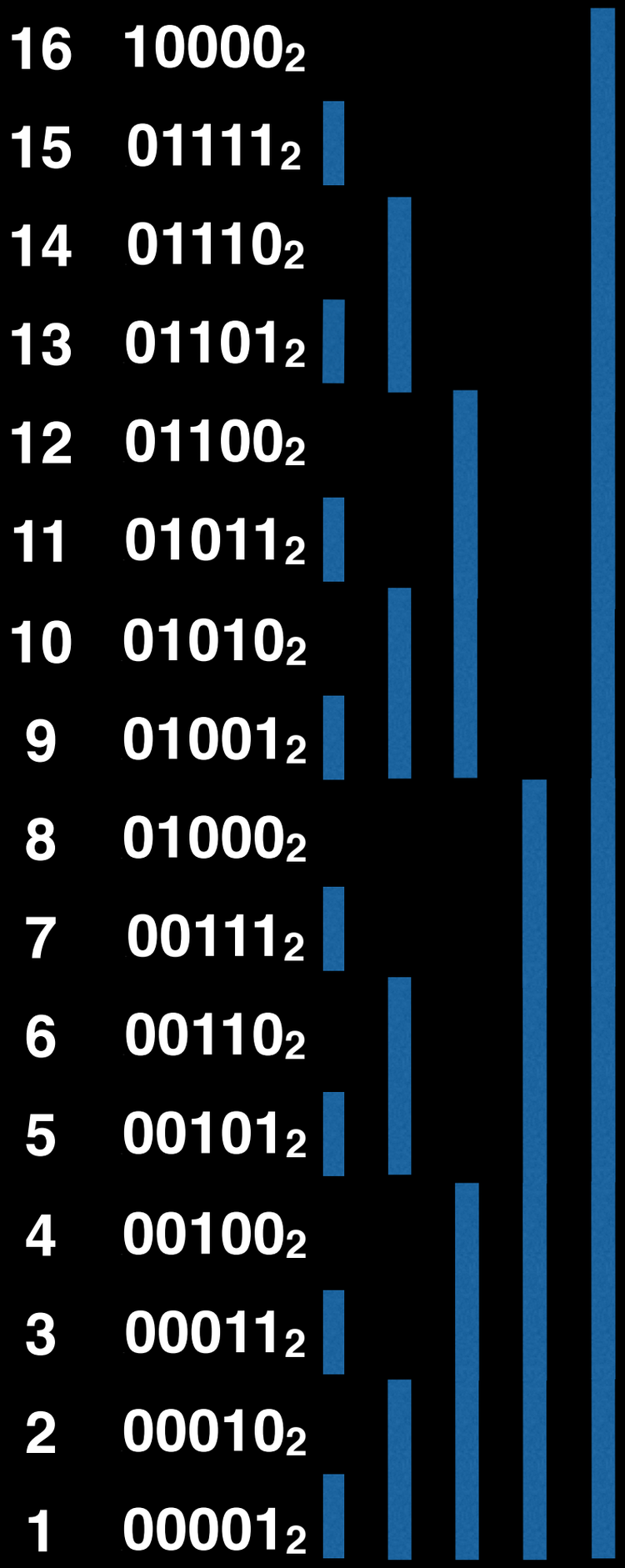
Instead of querying a range to find the interval sum we want to update a cell in our array.

Recall that with range queries we cascaded down from the current index by **continuously removing the LSB.**

$13 = 1101_2$, $1101_2 - 0001_2 = 1100_2$

$12 = 1100_2$, $1100_2 - 0100_2 = 1000_2$

$8 = 1000_2$



Point Updates

Instead of querying a range to find the interval sum we want to update a cell in our array.

Recall that with range queries we cascaded down from the current index by **continuously removing the LSB.**

$13 = 1101_2$, $1101_2 - 0001_2 = 1100_2$

↓

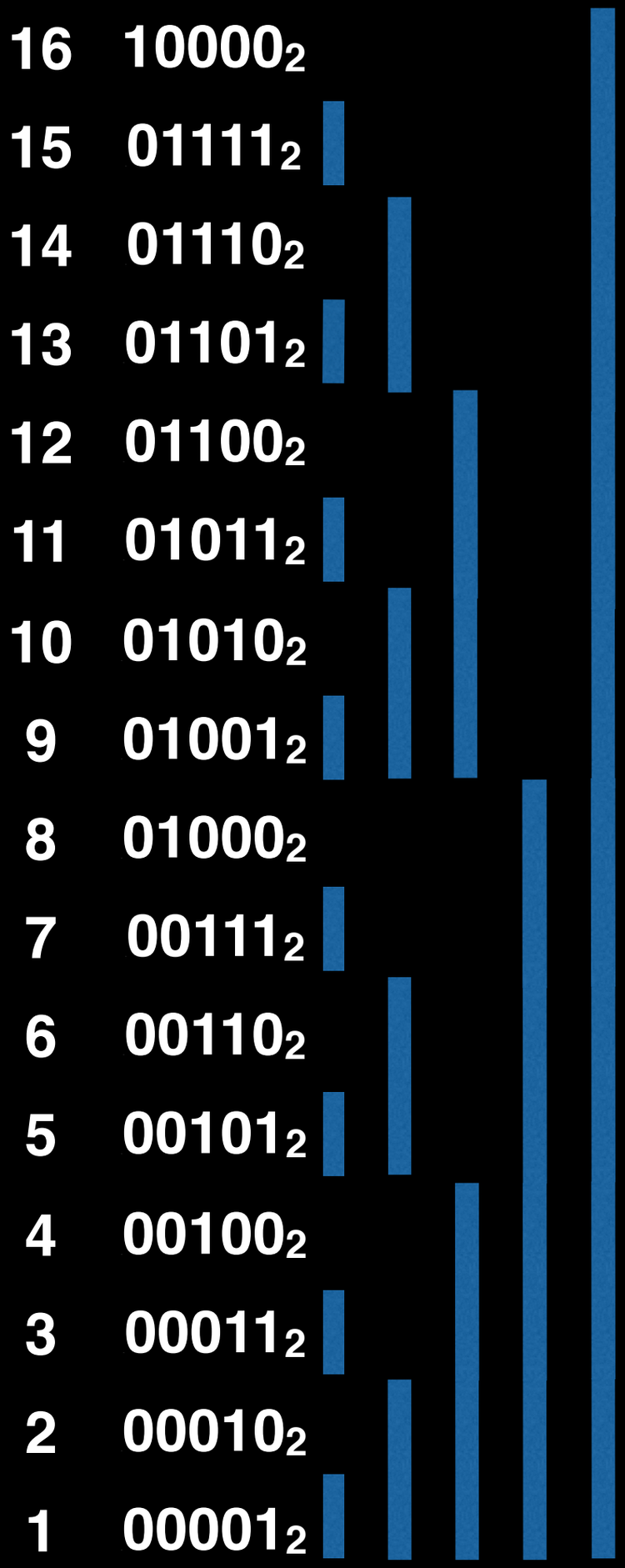
$12 = 1100_2$, $1100_2 - 0100_2 = 1000_2$

↓

$8 = 1000_2$, $1000_2 - 1000_2 = 0000_2$

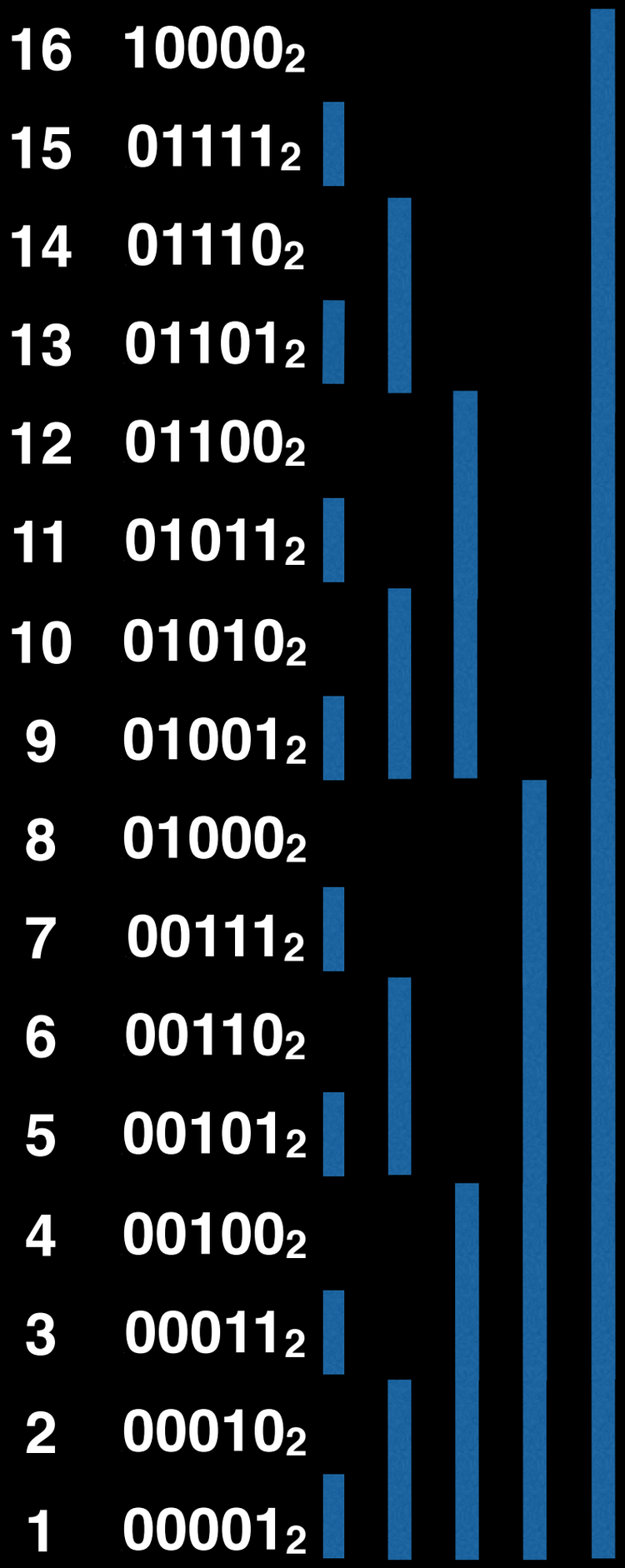
↓

$0 = 0000_2$



Point Updates

Point updates are the opposite of this, we want to **add the LSB** to propagate the value up to the cells responsible for us.



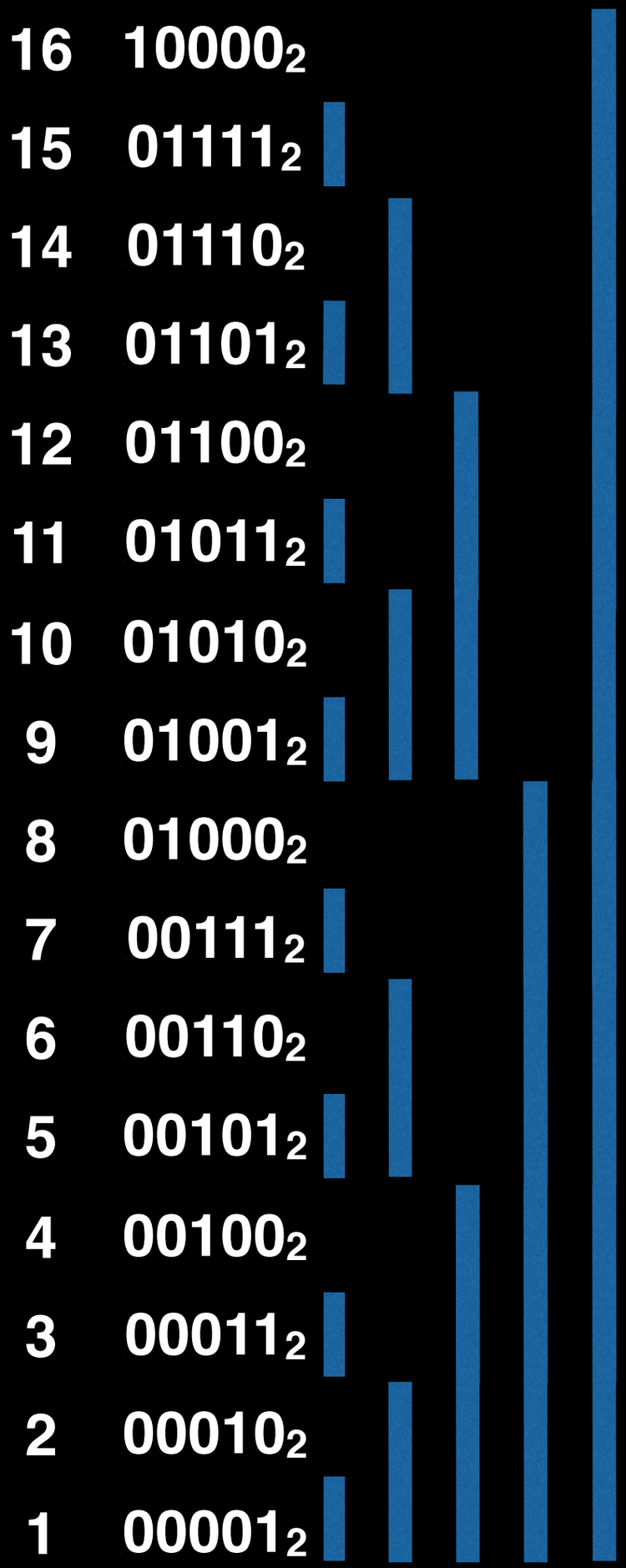
Point Updates

Point updates are the opposite of this, we want to **add the LSB** to propagate the value up to the cells responsible for us.

$9 = 1001_2$, $1001_2 + 0001_2 = 1010_2$

↓

$10 = 1010_2$



Point Updates

Point updates are the opposite of this, we want to **add the LSB** to propagate the value up to the cells responsible for us.

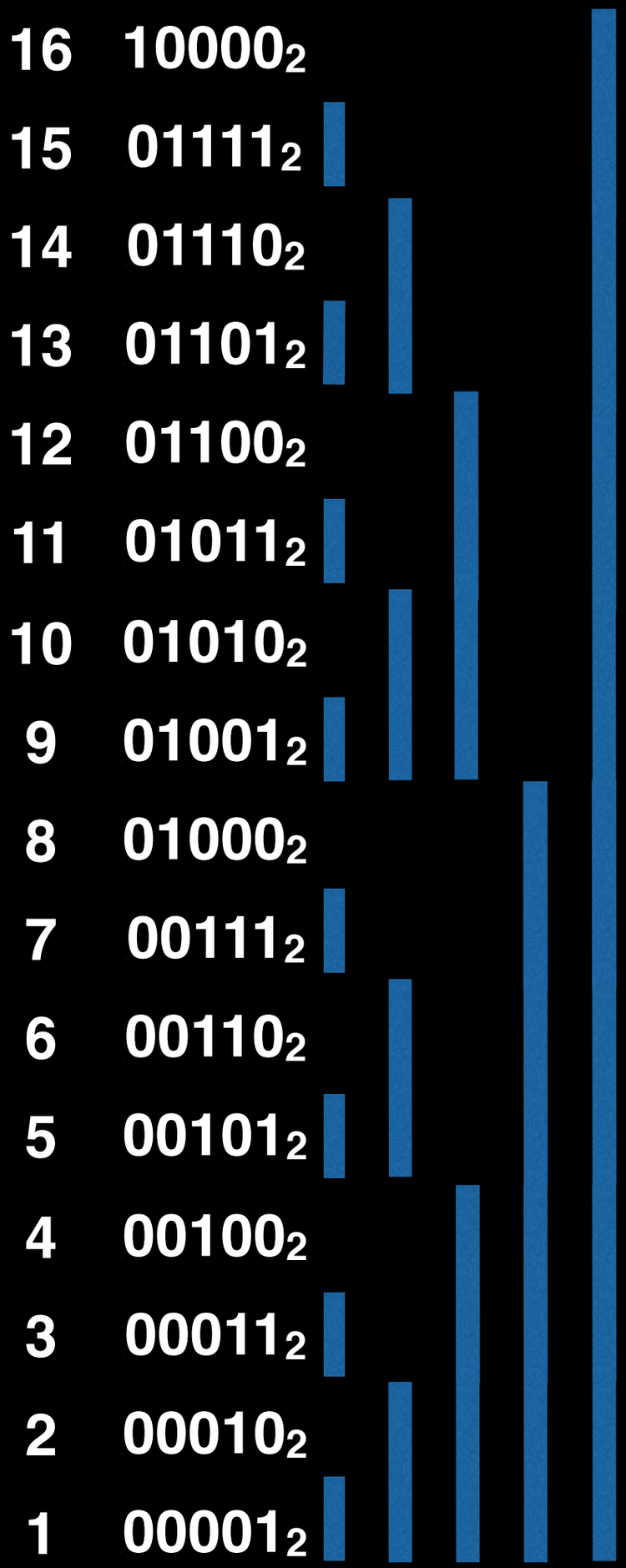
$9 = 1001_2$, $1001_2 + 0001_2 = 1010_2$

↓

$10 = 1010_2$, $1010_2 + 0010_2 = 1100_2$

↓

$12 = 1100_2$



Point Updates

Point updates are the opposite of this, we want to **add the LSB** to propagate the value up to the cells responsible for us.

$9 = 1001_2$, $1001_2 + 0001_2 = 1010_2$

↓

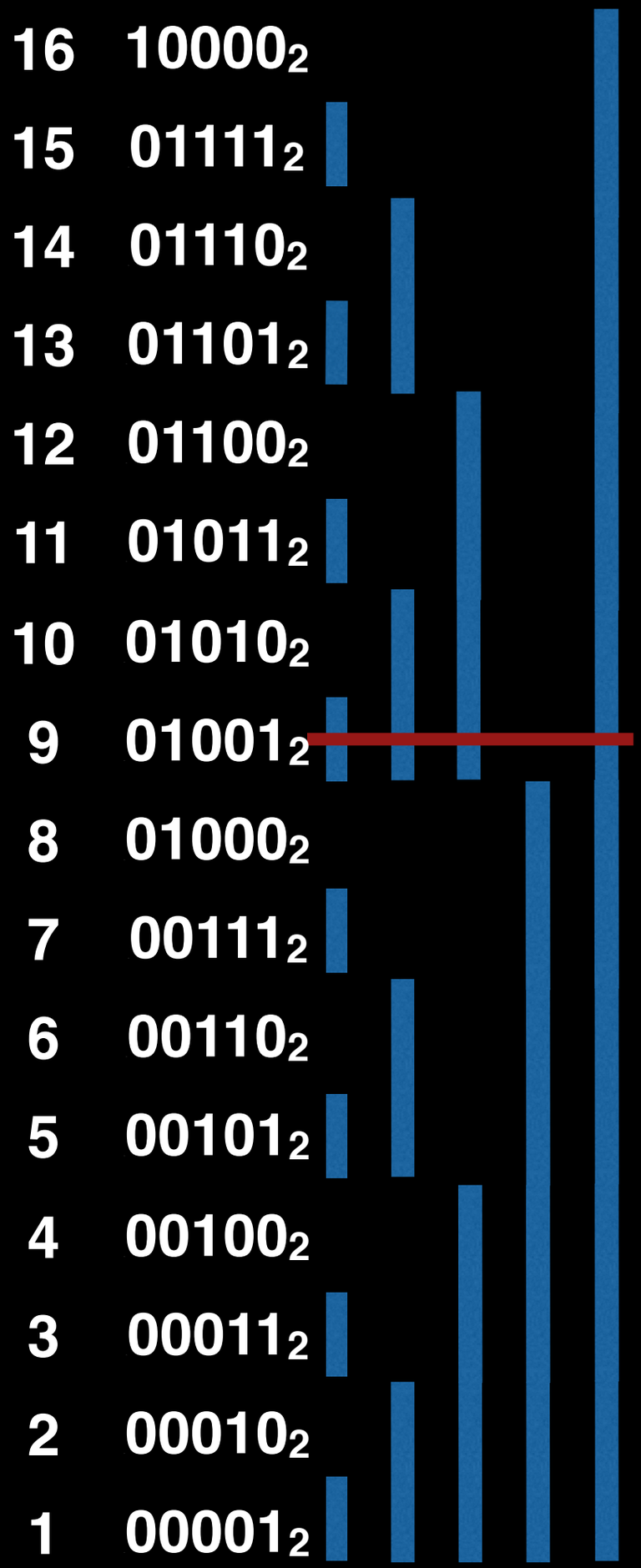
$10 = 1010_2$, $1010_2 + 0010_2 = 1100_2$

↓

$12 = 1100_2$, $1100_2 + 0100_2 = 10000_2$

↓

$16 = 10000_2$



Point Updates

Point updates are the opposite of this, we want to **add the LSB** to propagate the value up to the cells responsible for us.

$9 = 1001_2$, $1001_2 + 0001_2 = 1010_2$

↓

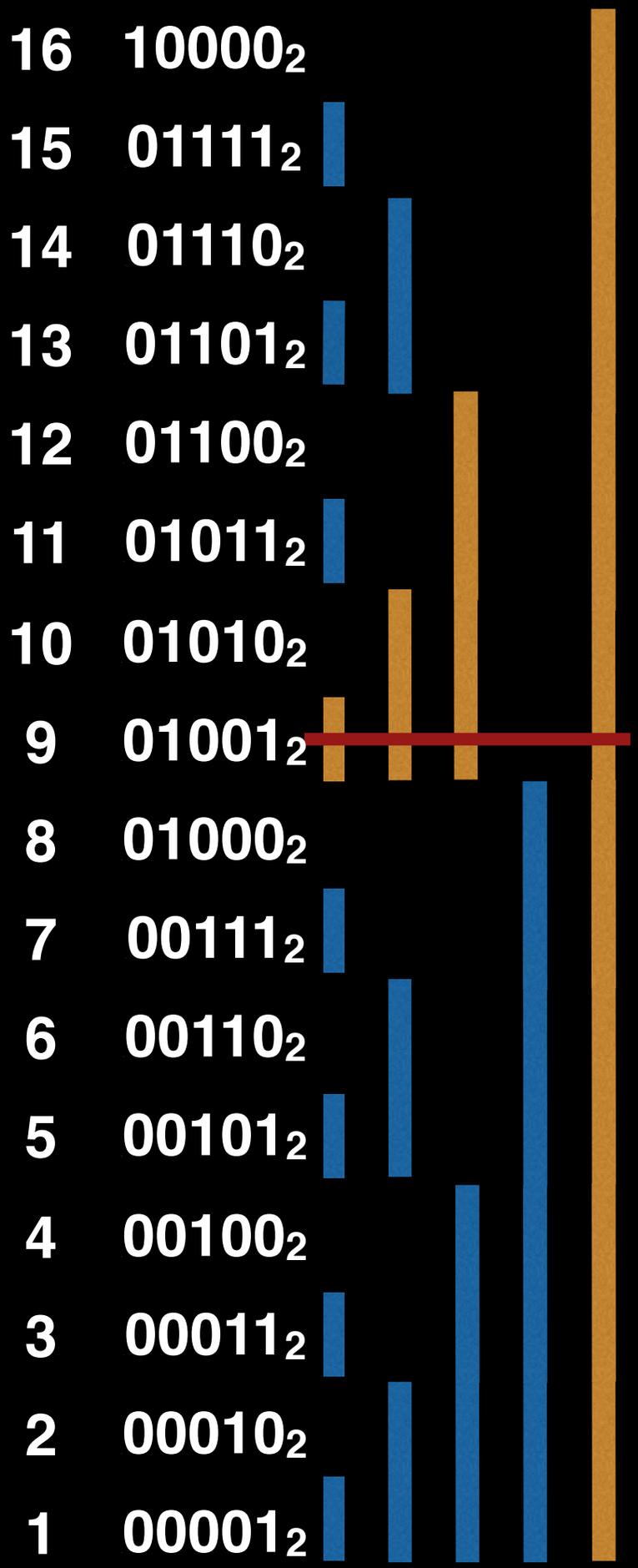
$10 = 1010_2$, $1010_2 + 0010_2 = 1100_2$

↓

$12 = 1100_2$, $1100_2 + 0100_2 = 10000_2$

↓

$16 = 10000_2$



Point Updates

Point updates are the opposite of this, we want to **add the LSB** to propagate the value up to the cells responsible for us.

$9 = 1001_2$, $1001_2 + 0001_2 = 1010_2$

↓

$10 = 1010_2$, $1010_2 + 0010_2 = 1100_2$

↓

$12 = 1100_2$, $1100_2 + 0100_2 = 10000_2$

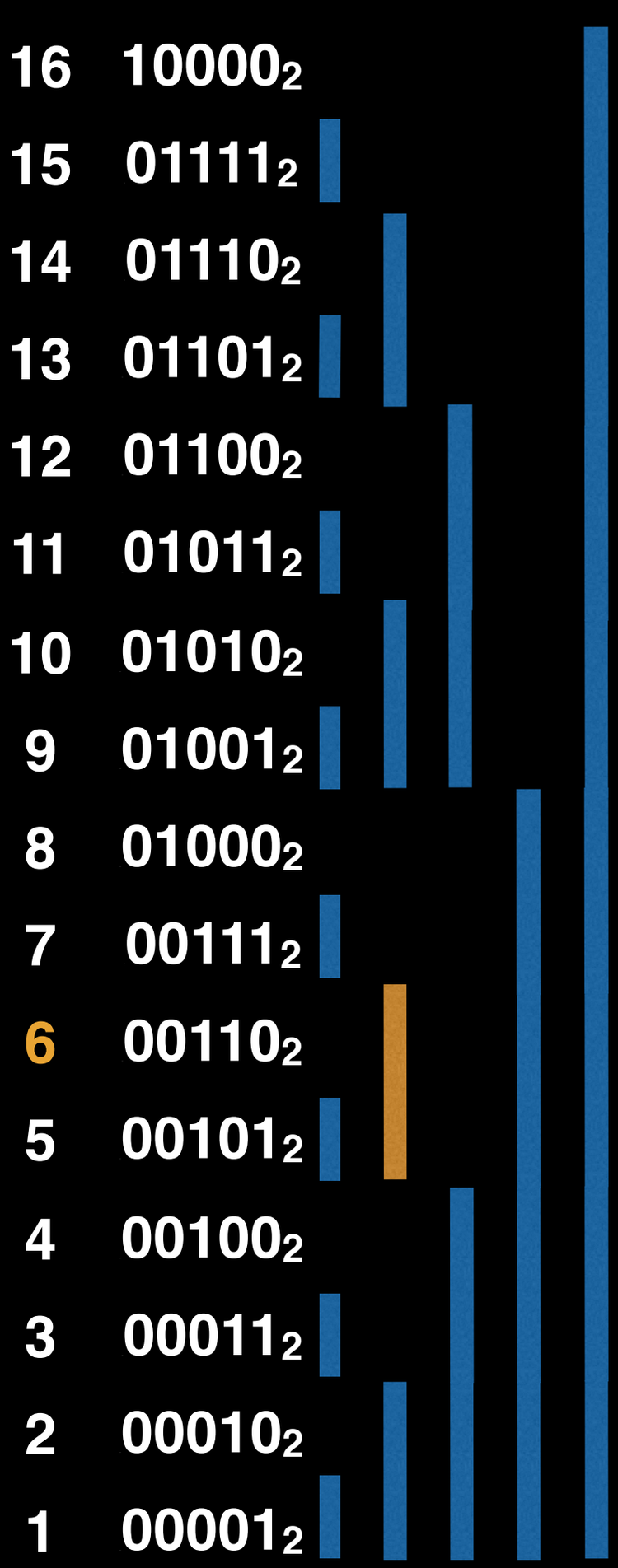
↓

$16 = 10000_2$

Point Updates

If we add x to position 6 in the Fenwick tree which cells do we also need to modify?

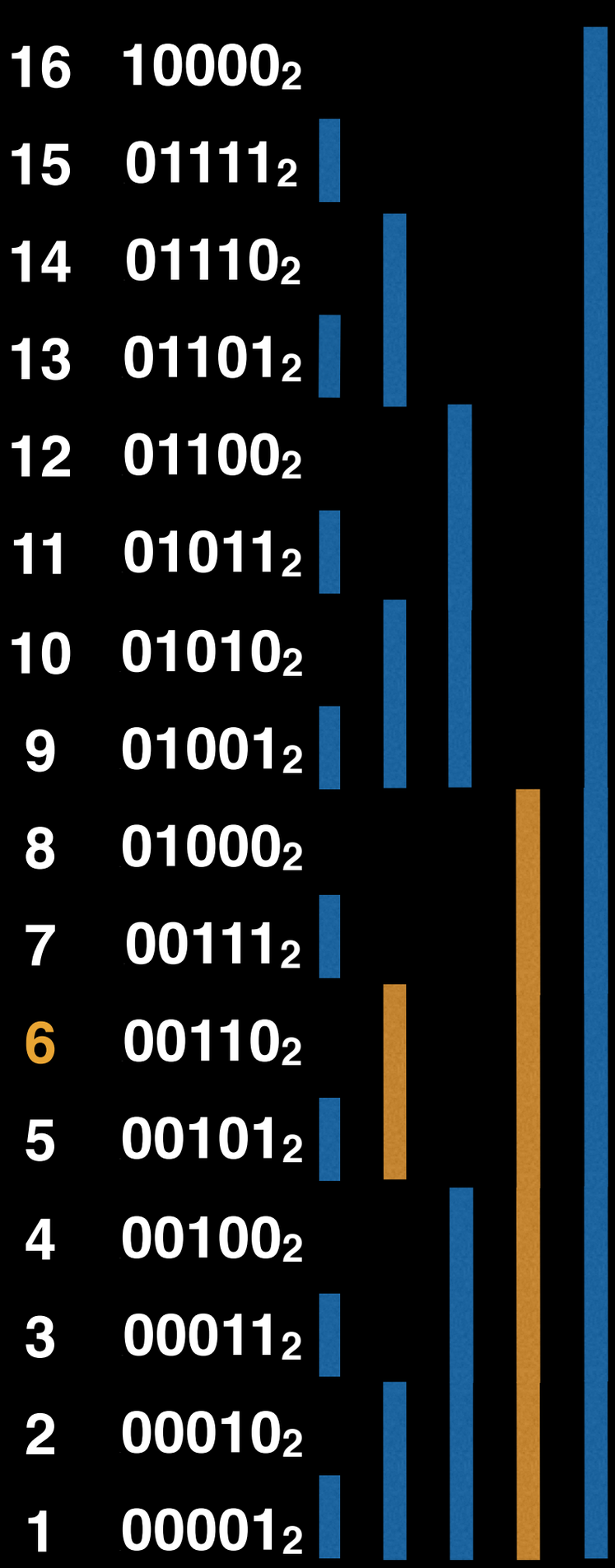
Point Updates



If we add x to position 6 in the Fenwick tree which cells do we also need to modify?

$6 = 0110_2$

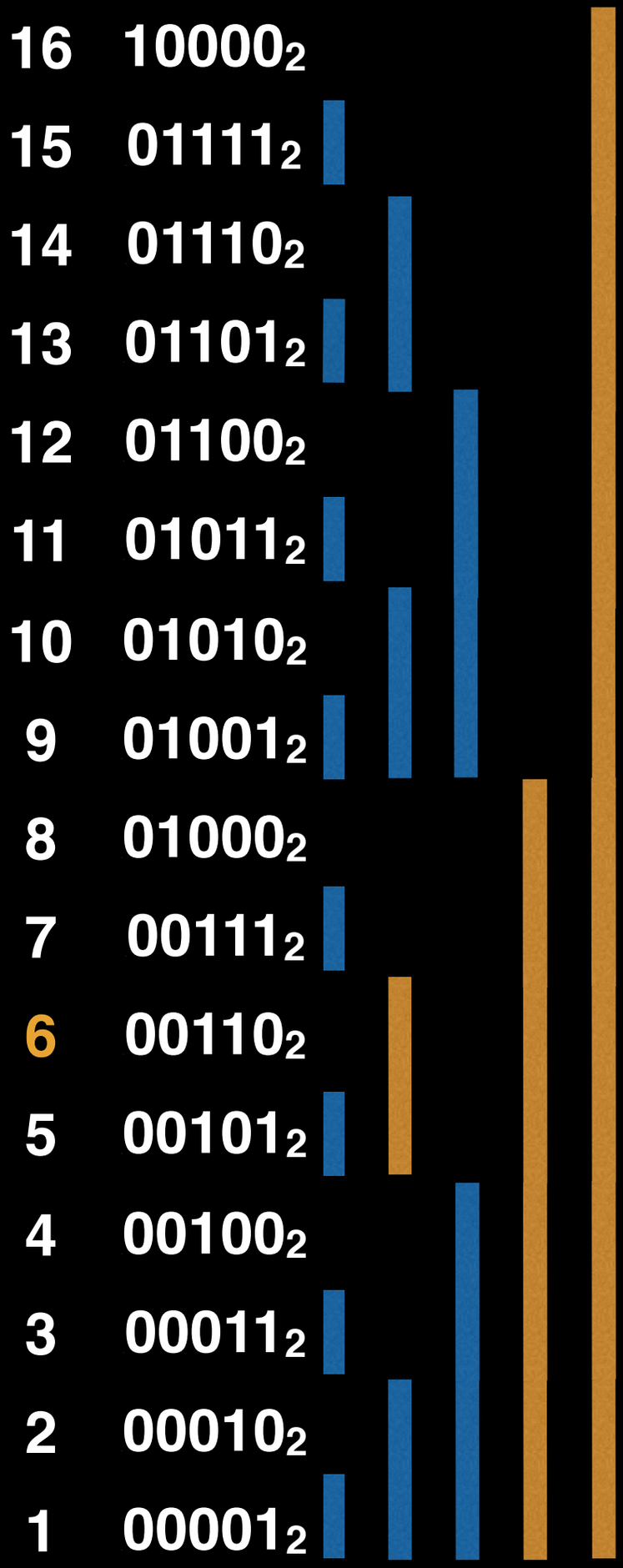
Point Updates



If we add x to position 6 in the Fenwick tree which cells do we also need to modify?

$6 = 0110_2$, $0110_2 + 0010_2 = 1000_2$

$8 = 1000_2$



Point Updates

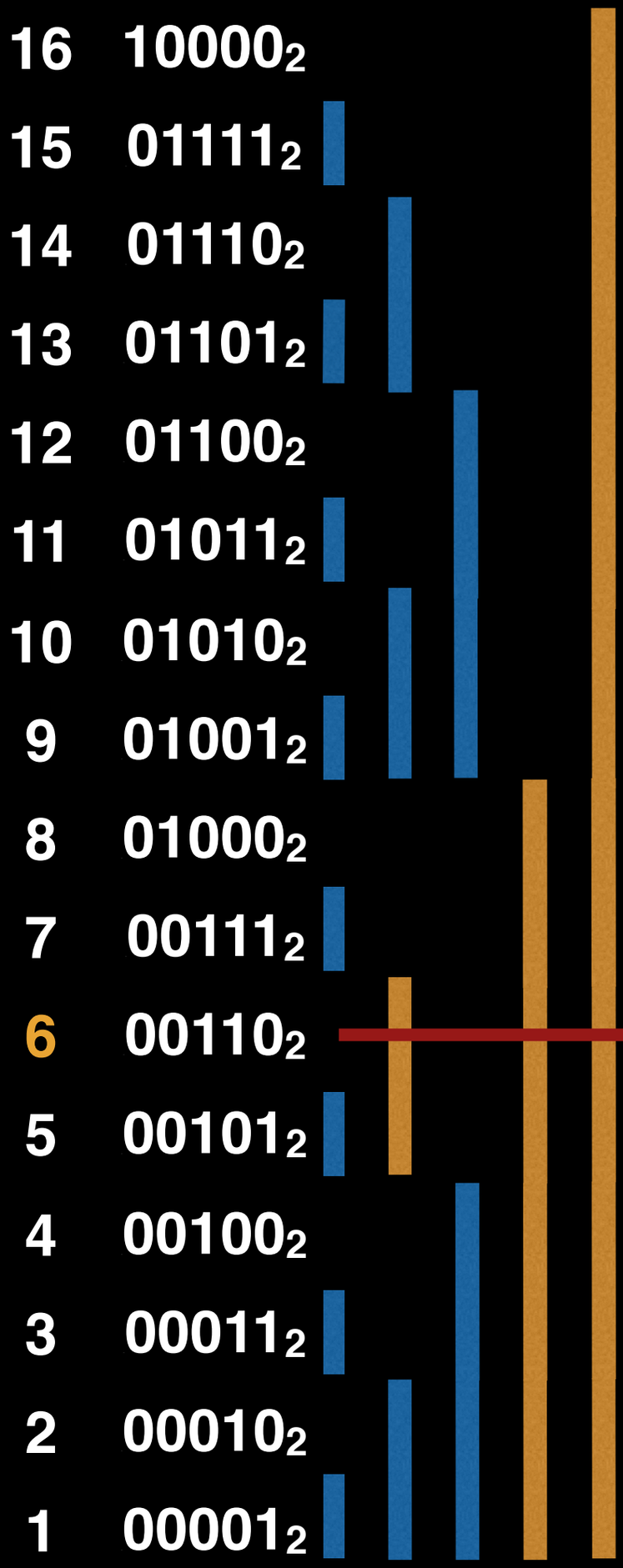
If we add x to position 6 in the Fenwick tree which cells do we also need to modify?

$6 = 0110_2$, $0110_2 + 0010_2 = 1000_2$

$8 = 1000_2$, $1000_2 + 1000_2 = 10000_2$

$16 = 10000_2$

Point Updates



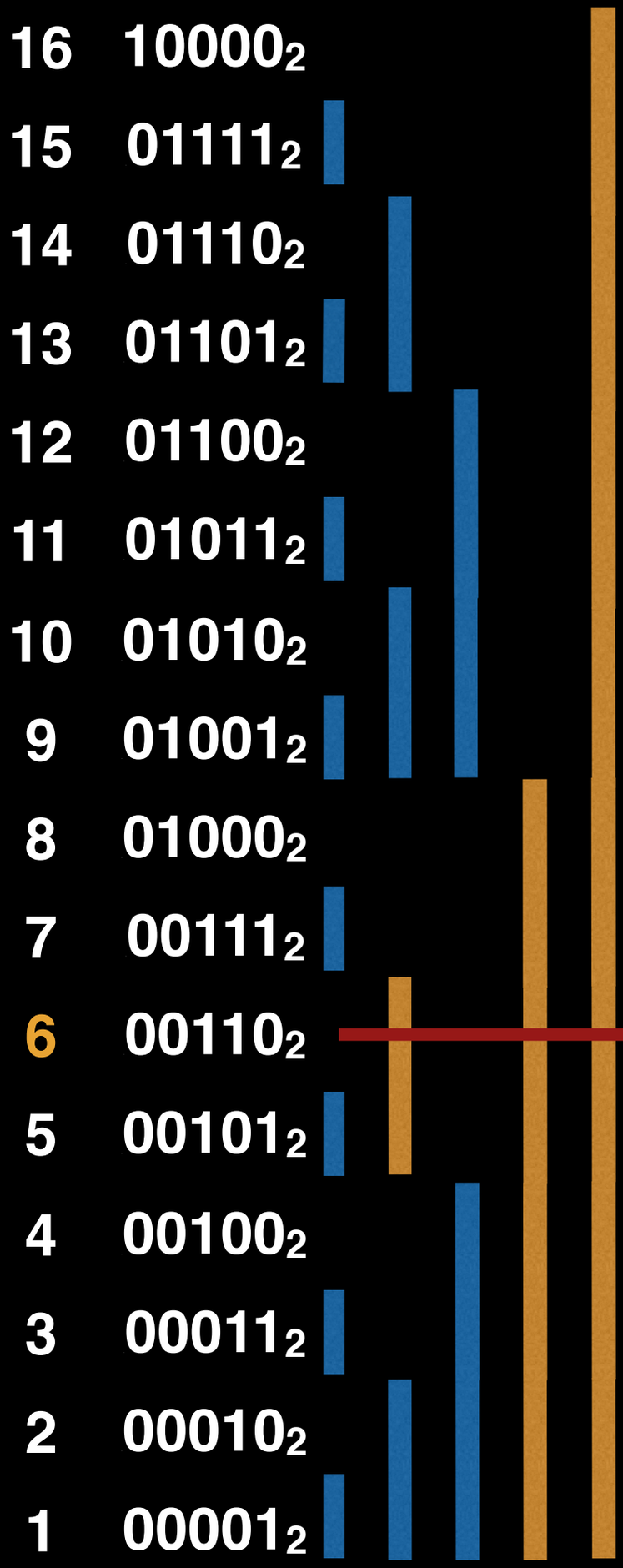
If we add x to position 6 in the Fenwick tree which cells do we also need to modify?

$6 = 0110_2$, $0110_2 + 0010_2 = 1000_2$

$8 = 1000_2$, $1000_2 + 1000_2 = 10000_2$

$16 = 10000_2$

Point Updates



If we add x to position 6 in the Fenwick tree which cells do we also need to modify?

$6 = 0110_2$, $0110_2 + 0010_2 = 1000_2$

$8 = 1000_2$, $1000_2 + 1000_2 = 10000_2$

$16 = 10000_2$

Required Updates:

A[6] = A[6] + x

A[8] = A[8] + x

A[16] = A[16] + x

Fenwick Tree Construction

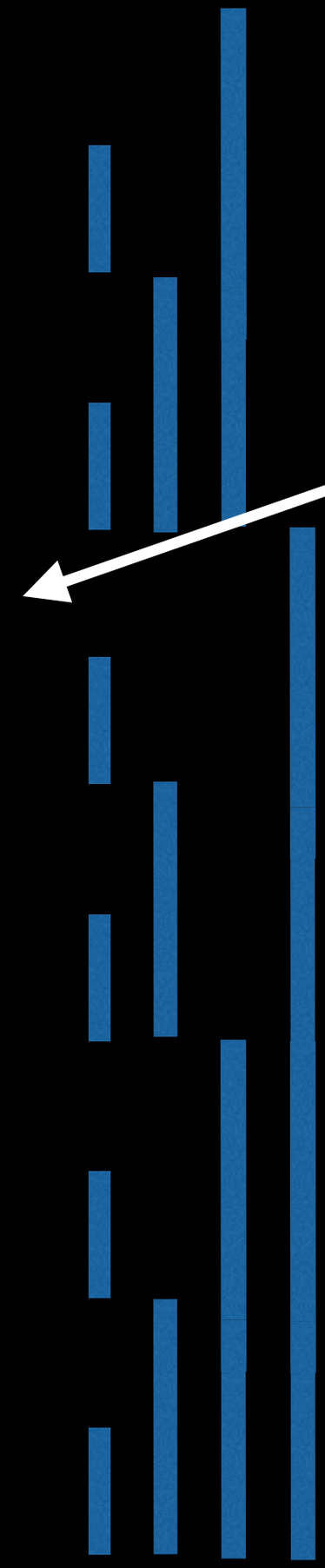
Naive Construction

Let A be an array of values. For each element in A at index *i* do a point update on the Fenwick tree with a value of A[i]. There are *n* elements and each point update takes O(log(n)) for a total of O(nlog(n)), can we do better?

Linear Construction

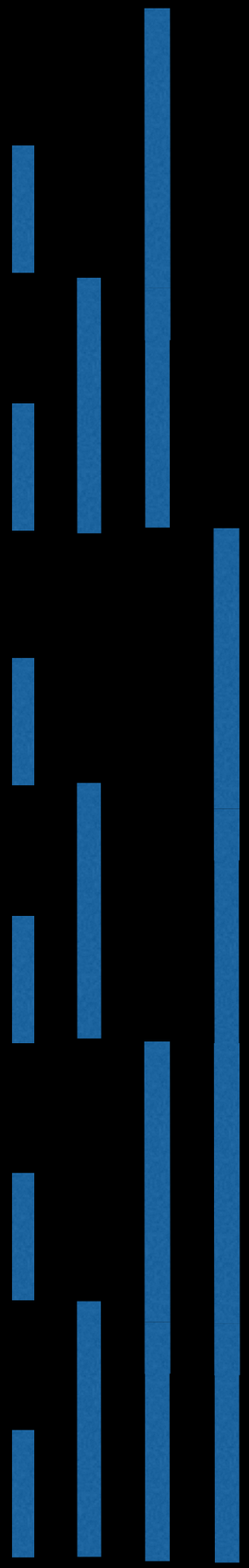
Input values we wish to turn into a legitimate Fenwick tree.

12	1100_2	-8
11	1011_2	4
10	1010_2	2
9	1001_2	-9
8	1000_2	-8
7	0111_2	5
6	0110_2	11
5	0101_2	3
4	0100_2	7
3	0011_2	-2
2	0010_2	4
1	0001_2	3



Linear Construction

12	1100_2	-8
11	1011_2	4
10	1010_2	2
9	1001_2	-9
8	1000_2	-8
7	0111_2	5
6	0110_2	11
5	0101_2	3
4	0100_2	7
3	0011_2	-2
2	0010_2	4
1	0001_2	3

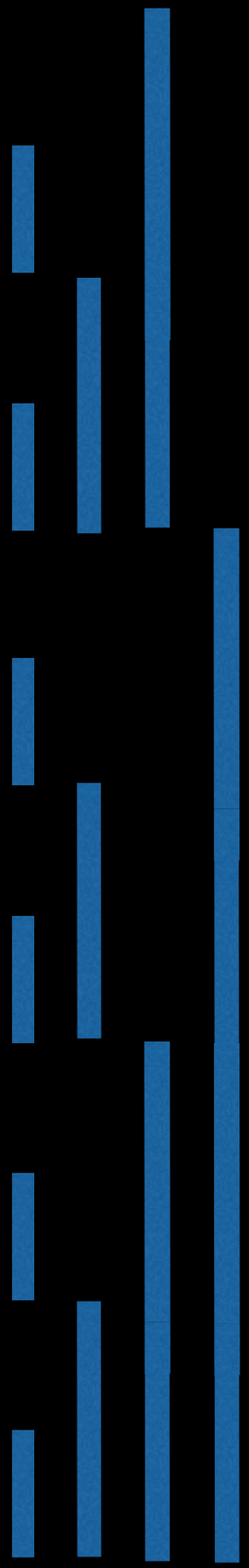


Idea: Add the value in the current cell to the **immediate cell** that is responsible for us. This resembles what we did for point updates but only one cell at a time.

This will make the ‘cascading’ effect in range queries possible by propagating the value in each cell throughout the tree.

Linear Construction

12	1100_2	-8
11	1011_2	4
10	1010_2	2
9	1001_2	-9
8	1000_2	-8
7	0111_2	5
6	0110_2	11
5	0101_2	3
4	0100_2	7
3	0011_2	-2
2	0010_2	4
1	0001_2	3



Let i be the current index

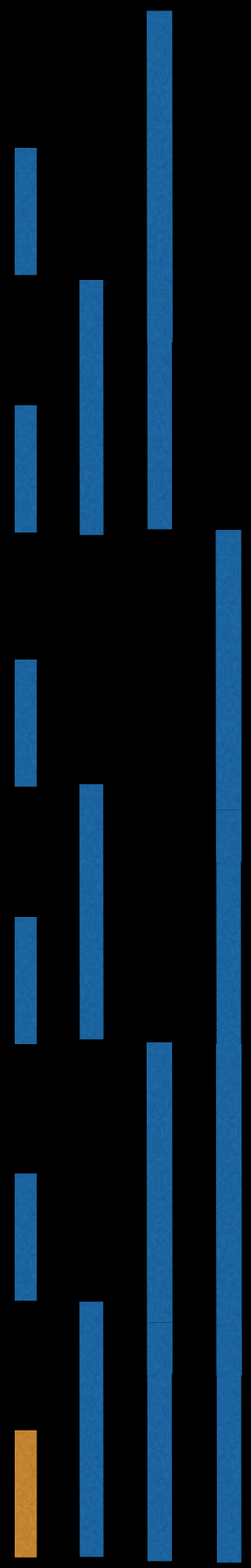
The immediate cell above us is at position j given by:

$$j := i + LSB(i)$$

Where LSB is the **Least Significant Bit** of i

Linear Construction

12	1100_2	-8
11	1011_2	4
10	1010_2	2
9	1001_2	-9
8	1000_2	-8
7	0111_2	5
6	0110_2	11
5	0101_2	3
4	0100_2	7
3	0011_2	-2
2	0010_2	4
1	0001_2	3

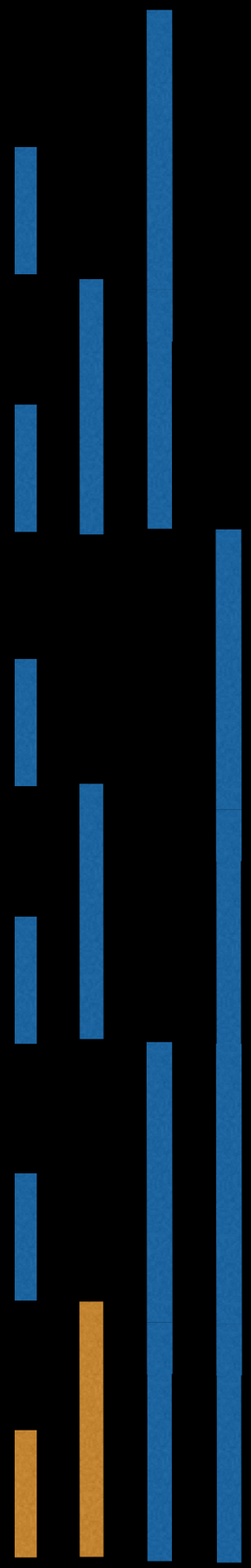


$i = 1 = 0001_2$

$j = 0001_2 + 0001_2 = 0010_2$
$= 2$

Linear Construction

12	1100_2	-8
11	1011_2	4
10	1010_2	2
9	1001_2	-9
8	1000_2	-8
7	0111_2	5
6	0110_2	11
5	0101_2	3
4	0100_2	7
3	0011_2	-2
2	0010_2	4+3
1	0001_2	3

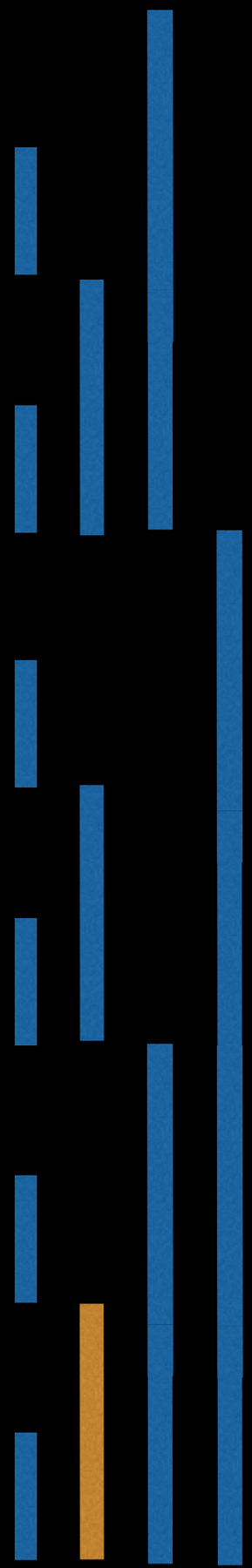


$i = 1 = 0001_2$

$j = 0001_2 + 0001_2 = 0010_2$
$= 2$

Linear Construction

12	1100_2	-8
11	1011_2	4
10	1010_2	2
9	1001_2	-9
8	1000_2	-8
7	0111_2	5
6	0110_2	11
5	0101_2	3
4	0100_2	7
3	0011_2	-2
2	0010_2	7
1	0001_2	3

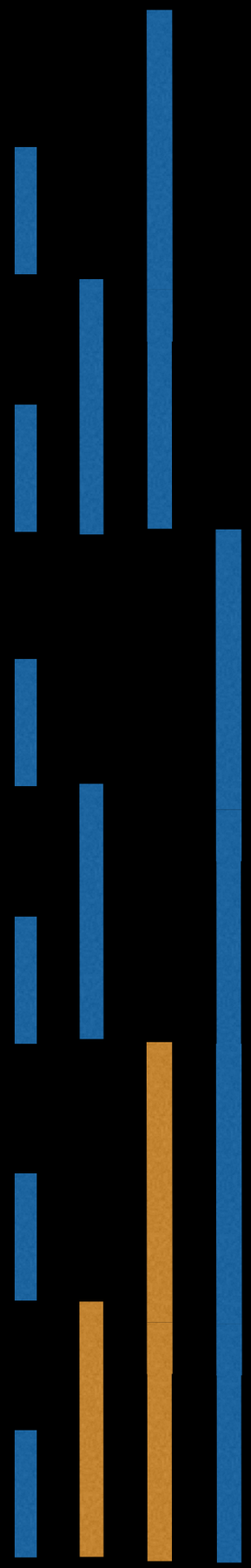


$i = 2 = 0010_2$

$j = 0010_2 + 0010_2 = 0100_2$
$= 4$

Linear Construction

12	1100_2	-8
11	1011_2	4
10	1010_2	2
9	1001_2	-9
8	1000_2	-8
7	0111_2	5
6	0110_2	11
5	0101_2	3
4	0100_2	7+7
3	0011_2	-2
2	0010_2	7
1	0001_2	3

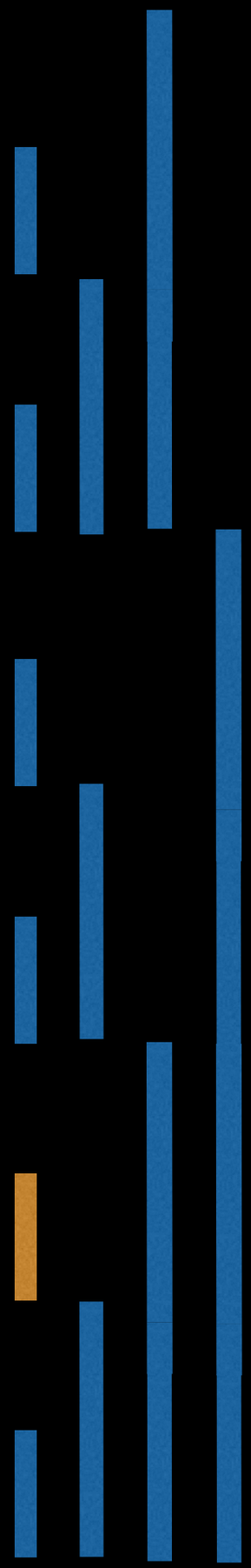


$i = 2 = 0010_2$

$j = 0010_2 + 0010_2 = 0100_2$
$= 4$

Linear Construction

12	1100_2	-8
11	1011_2	4
10	1010_2	2
9	1001_2	-9
8	1000_2	-8
7	0111_2	5
6	0110_2	11
5	0101_2	3
4	0100_2	14
3	0011_2	-2
2	0010_2	7
1	0001_2	3

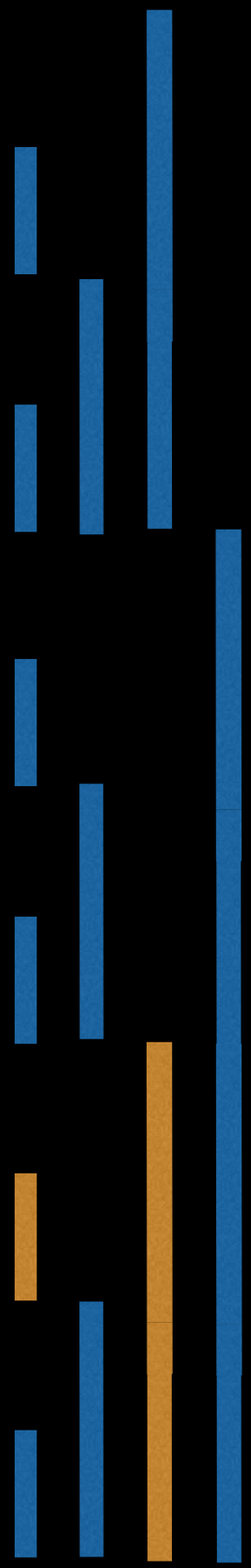


$i = 3 = 0011_2$

$j = 0011_2 + 0001_2 = 0100_2$
$= 4$

Linear Construction

12	1100_2	-8
11	1011_2	4
10	1010_2	2
9	1001_2	-9
8	1000_2	-8
7	0111_2	5
6	0110_2	11
5	0101_2	3
4	0100_2	14+-2
3	0011_2	-2
2	0010_2	7
1	0001_2	3

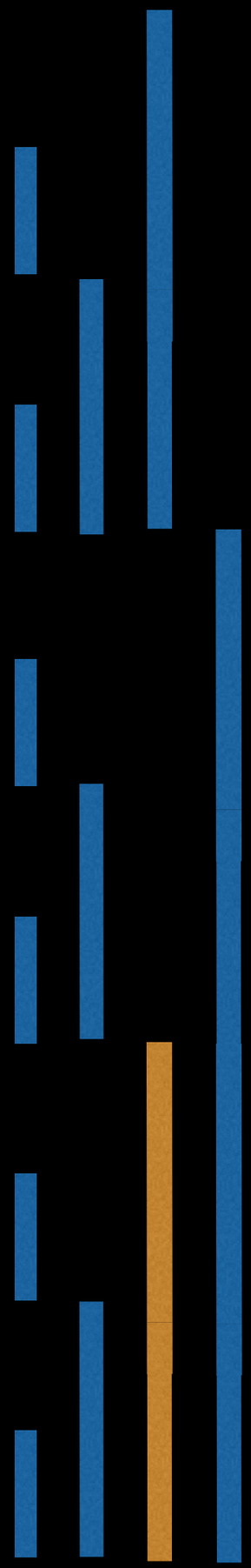


$i = 3 = 0011_2$

$j = 0011_2 + 0001_2 = 0100_2$
$= 4$

Linear Construction

12	1100_2	-8
11	1011_2	4
10	1010_2	2
9	1001_2	-9
8	1000_2	-8
7	0111_2	5
6	0110_2	11
5	0101_2	3
4	0100_2	12
3	0011_2	-2
2	0010_2	7
1	0001_2	3

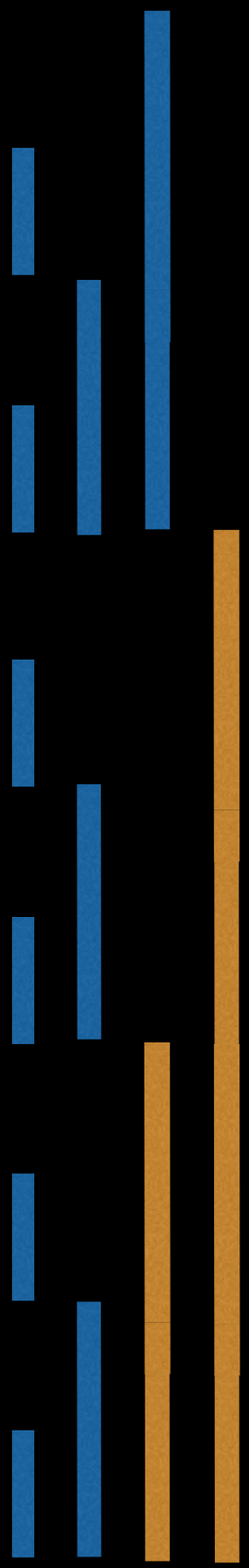


$i = 4 = 0100_2$

$j = 0100_2 + 0100_2 = 1000_2$
$= 8$

Linear Construction

12	1100_2	-8
11	1011_2	4
10	1010_2	2
9	1001_2	-9
8	1000_2	-8+12
7	0111_2	5
6	0110_2	11
5	0101_2	3
4	0100_2	12
3	0011_2	-2
2	0010_2	7
1	0001_2	3

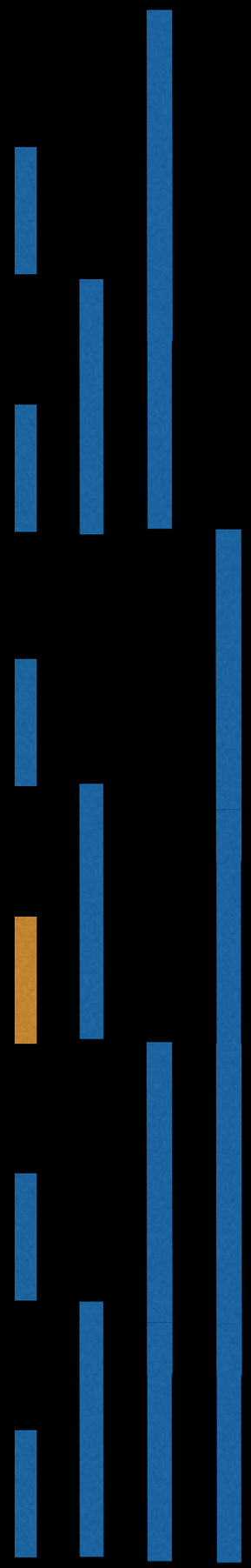


$i = 4 = 0100_2$

$j = 0100_2 + 0100_2 = 1000_2$
$= 8$

Linear Construction

12	1100_2	-8
11	1011_2	4
10	1010_2	2
9	1001_2	-9
8	1000_2	4
7	0111_2	5
6	0110_2	11
5	0101_2	3
4	0100_2	12
3	0011_2	-2
2	0010_2	7
1	0001_2	3

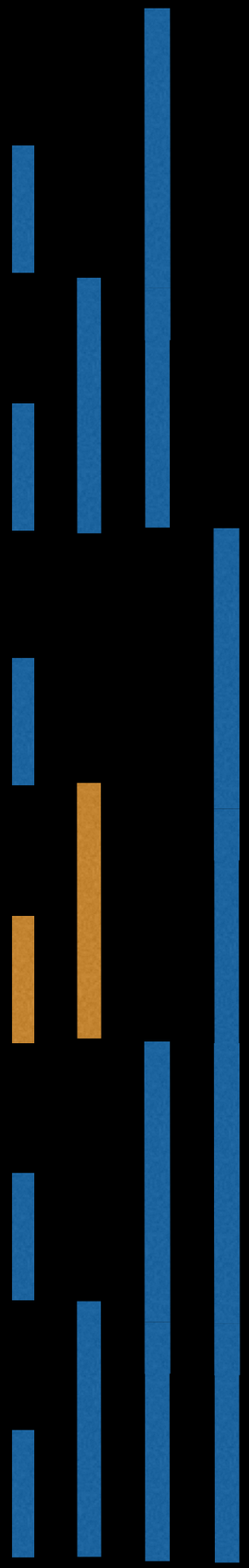


$i = 5 = 0101_2$

$j = 0101_2 + 0001_2 = 0110_2$
$= 6$

Linear Construction

12	1100_2	-8
11	1011_2	4
10	1010_2	2
9	1001_2	-9
8	1000_2	4
7	0111_2	5
6	0110_2	11+3
5	0101_2	3
4	0100_2	12
3	0011_2	-2
2	0010_2	7
1	0001_2	3

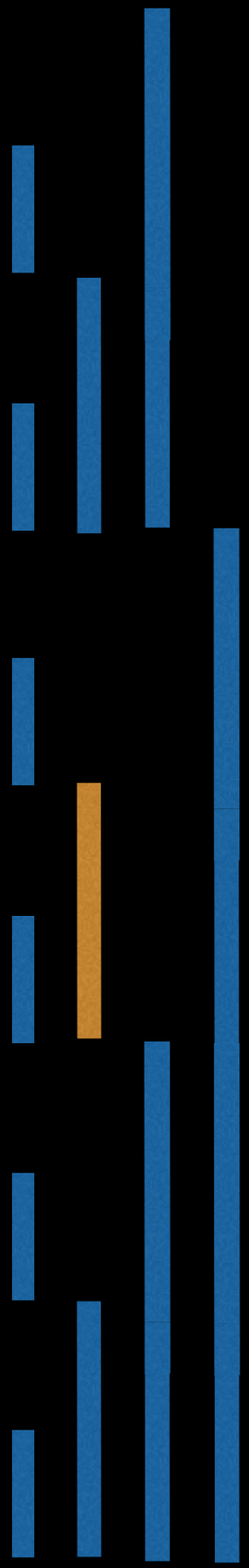


$i = 5 = 0101_2$

$j = 0101_2 + 0001_2 = 0110_2$
$= 6$

Linear Construction

12	1100_2	-8
11	1011_2	4
10	1010_2	2
9	1001_2	-9
8	1000_2	4
7	0111_2	5
6	0110_2	14
5	0101_2	3
4	0100_2	12
3	0011_2	-2
2	0010_2	7
1	0001_2	3

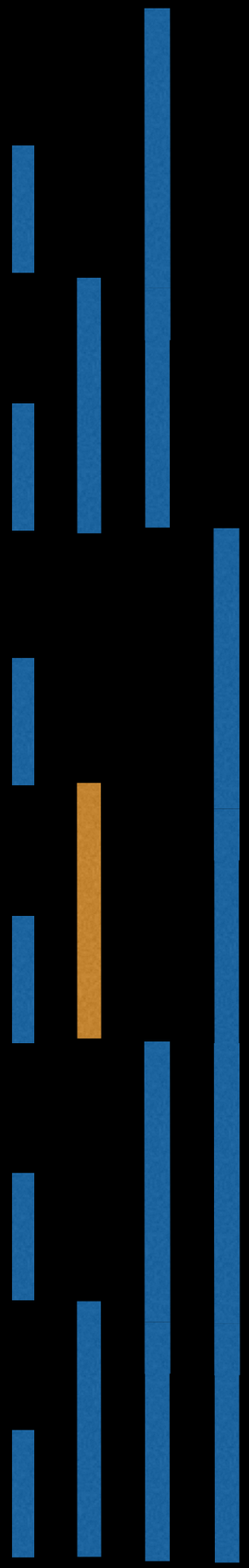


$i = 6 = 0110_2$

$j = 0110_2 + 0010_2 = 1000_2$
$= 8$

Linear Construction

12	1100_2	-8
11	1011_2	4
10	1010_2	2
9	1001_2	-9
8	1000_2	4+14
7	0111_2	5
6	0110_2	14
5	0101_2	3
4	0100_2	12
3	0011_2	-2
2	0010_2	7
1	0001_2	3

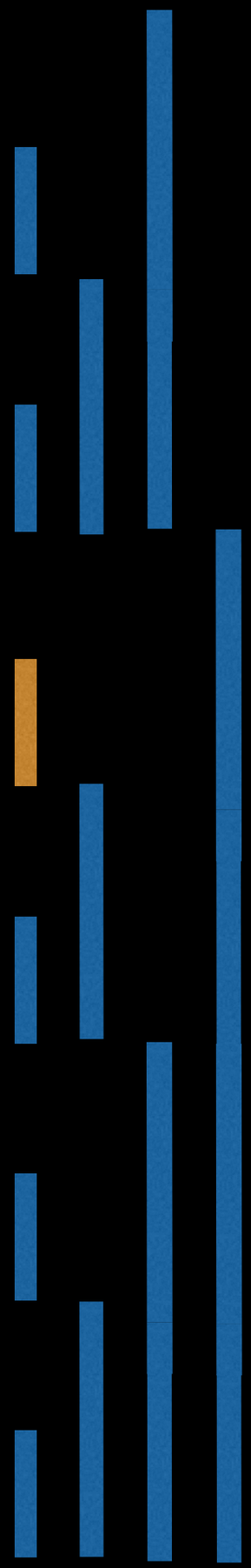


$i = 6 = 0110_2$

$j = 0110_2 + 0010_2 = 1000_2$
$= 8$

Linear Construction

12	1100_2	-8
11	1011_2	4
10	1010_2	2
9	1001_2	-9
8	1000_2	18
7	0111_2	5
6	0110_2	14
5	0101_2	3
4	0100_2	12
3	0011_2	-2
2	0010_2	7
1	0001_2	3

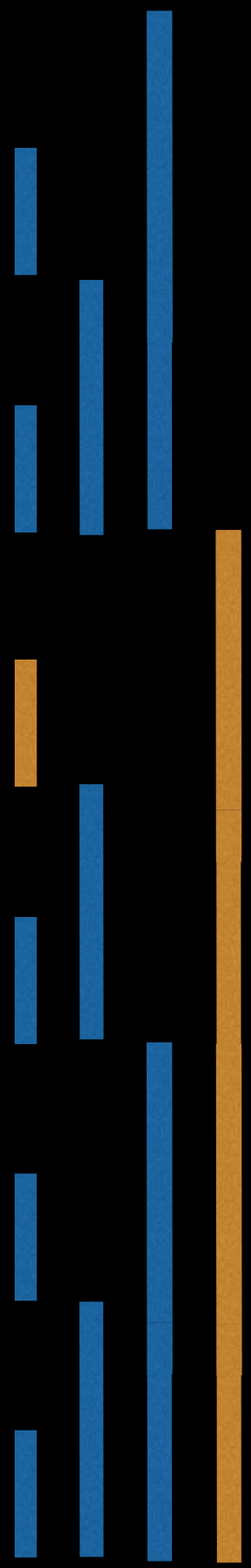


$i = 7 = 0111_2$

$j = 0111_2 + 0001_2 = 1000_2$
$= 8$

Linear Construction

12	1100_2	-8
11	1011_2	4
10	1010_2	2
9	1001_2	-9
8	1000_2	18+5
7	0111_2	5
6	0110_2	14
5	0101_2	3
4	0100_2	12
3	0011_2	-2
2	0010_2	7
1	0001_2	3

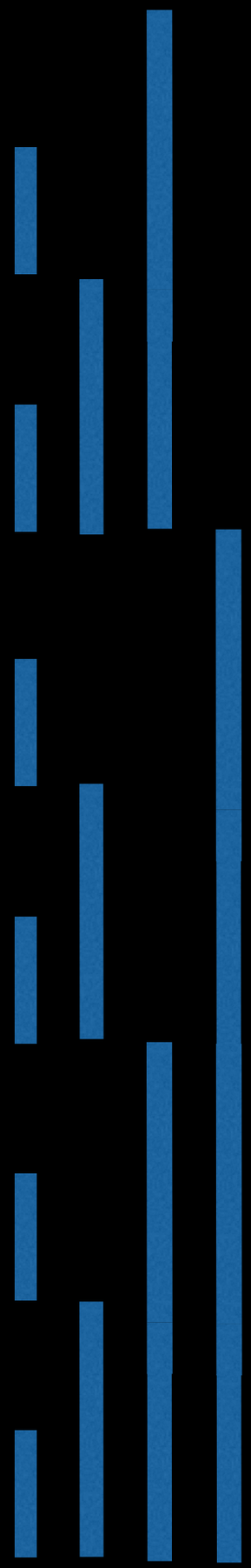


$i = 7 = 0111_2$

$j = 0111_2 + 0001_2 = 1000_2$
$= 8$

Linear Construction

12	1100_2	-8
11	1011_2	4
10	1010_2	2
9	1001_2	-9
8	1000_2	23
7	0111_2	5
6	0110_2	14
5	0101_2	3
4	0100_2	12
3	0011_2	-2
2	0010_2	7
1	0001_2	3



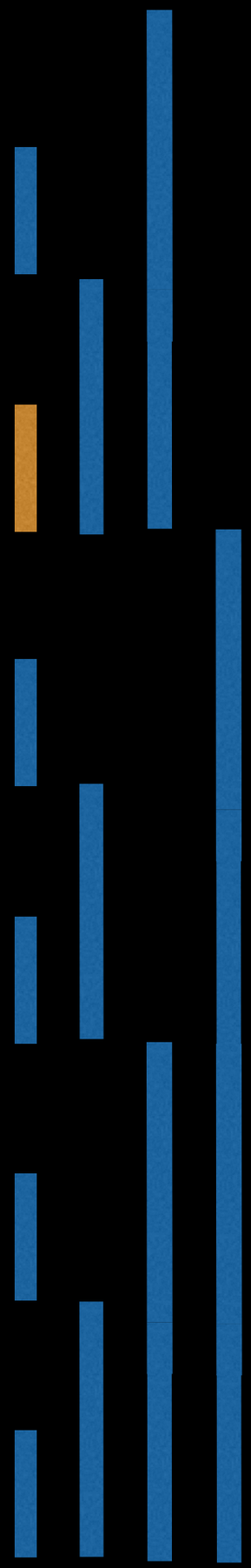
$i = 8 = 1000_2$

$j = 1000_2 + 1000_2 = 10000_2$
$= 16$

Ignore updating j if index is out of bounds

Linear Construction

12	1100_2	-8
11	1011_2	4
10	1010_2	2
9	1001_2	-9
8	1000_2	23
7	0111_2	5
6	0110_2	14
5	0101_2	3
4	0100_2	12
3	0011_2	-2
2	0010_2	7
1	0001_2	3

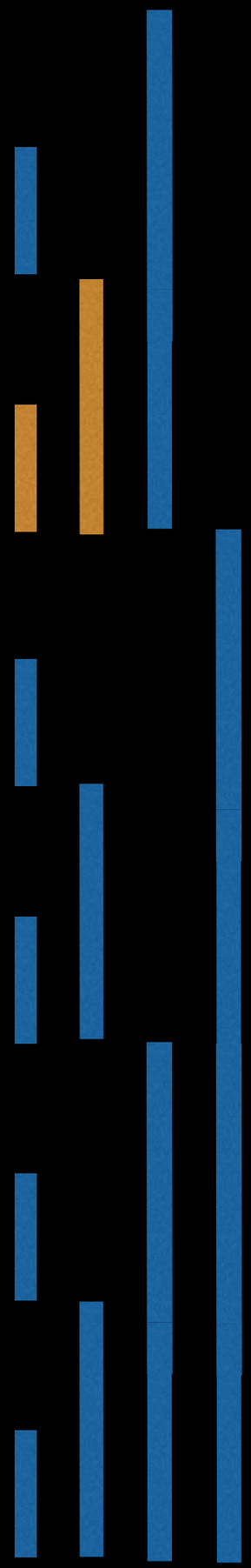


$i = 9 = 1001_2$

$j = 1001_2 + 0001_2 = 1010_2$
$= 10$

Linear Construction

12	1100_2	-8
11	1011_2	4
10	1010_2	2+-9
9	1001_2	-9
8	1000_2	23
7	0111_2	5
6	0110_2	14
5	0101_2	3
4	0100_2	12
3	0011_2	-2
2	0010_2	7
1	0001_2	3

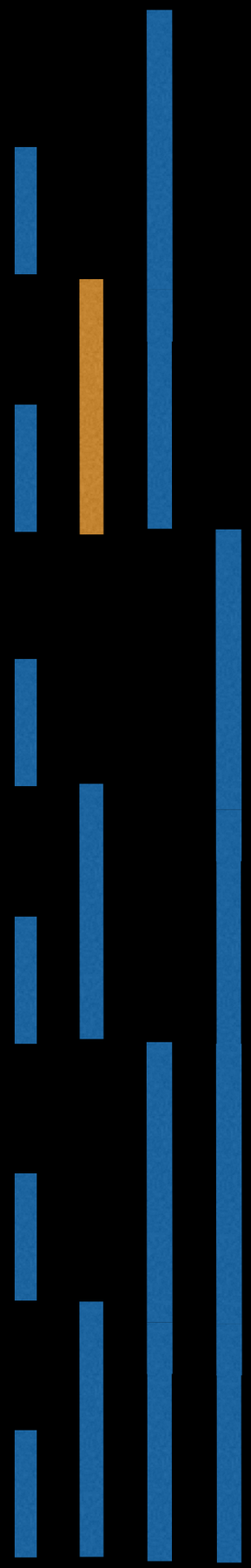


$i = 9 = 1001_2$

$j = 1001_2 + 0001_2 = 1010_2$
$= 10$

Linear Construction

12	1100_2	-8
11	1011_2	4
10	1010_2	-7
9	1001_2	-9
8	1000_2	23
7	0111_2	5
6	0110_2	14
5	0101_2	3
4	0100_2	12
3	0011_2	-2
2	0010_2	7
1	0001_2	3

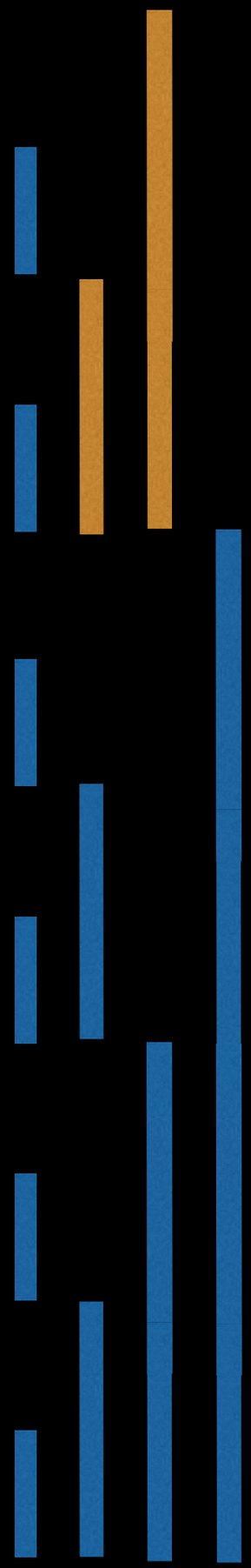


$i = 10 = 1010_2$

$j = 1010_2 + 0010_2 = 1100_2$
$= 12$

Linear Construction

12	1100_2	-8+-7
11	1011_2	4
10	1010_2	-7
9	1001_2	-9
8	1000_2	23
7	0111_2	5
6	0110_2	14
5	0101_2	3
4	0100_2	12
3	0011_2	-2
2	0010_2	7
1	0001_2	3

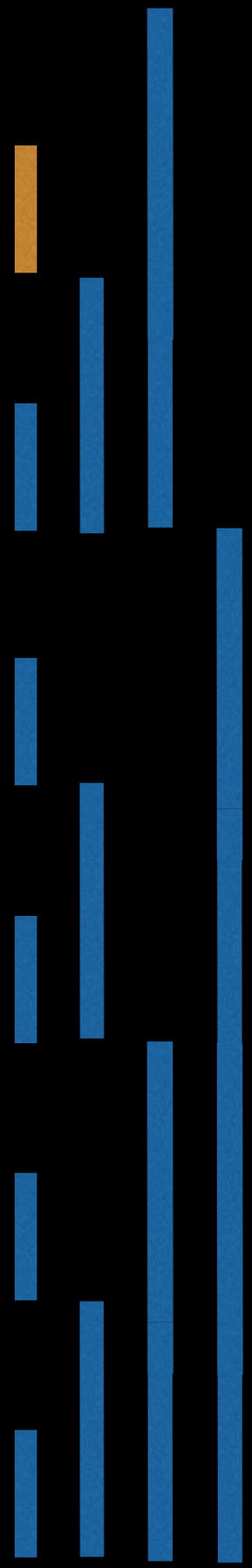


$i = 10 = 1010_2$

$j = 1010_2 + 0010_2 = 1100_2$
$= 12$

Linear Construction

12	1100_2	-15
11	1011_2	4
10	1010_2	-7
9	1001_2	-9
8	1000_2	23
7	0111_2	5
6	0110_2	14
5	0101_2	3
4	0100_2	12
3	0011_2	-2
2	0010_2	7
1	0001_2	3

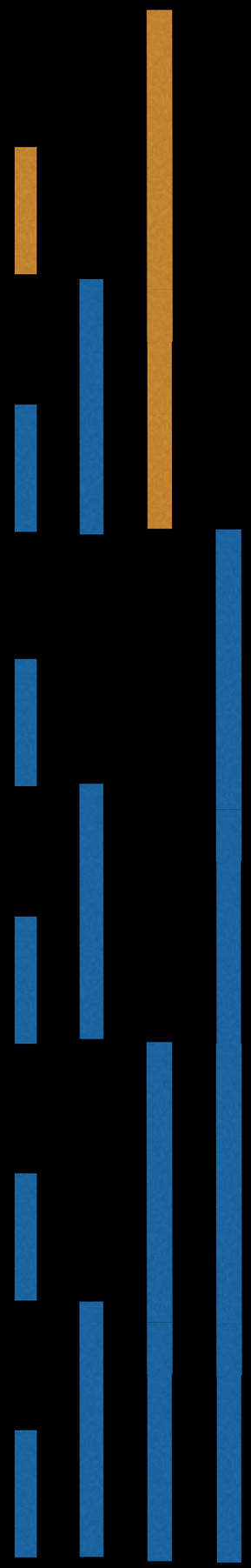


$i = 11 = 1011_2$

$j = 1011_2 + 0001_2 = 1100_2$
$= 12$

Linear Construction

12	1100_2	-15+4
11	1011_2	4
10	1010_2	-7
9	1001_2	-9
8	1000_2	23
7	0111_2	5
6	0110_2	14
5	0101_2	3
4	0100_2	12
3	0011_2	-2
2	0010_2	7
1	0001_2	3

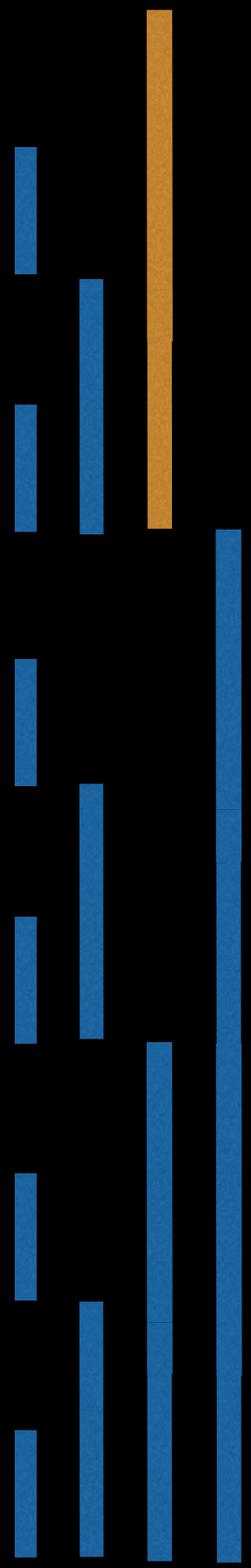


$i = 11 = 1011_2$

$j = 1011_2 + 0001_2 = 1100_2$
$= 12$

Linear Construction

12	1100_2	-11
11	1011_2	4
10	1010_2	-7
9	1001_2	-9
8	1000_2	23
7	0111_2	5
6	0110_2	14
5	0101_2	3
4	0100_2	12
3	0011_2	-2
2	0010_2	7
1	0001_2	3



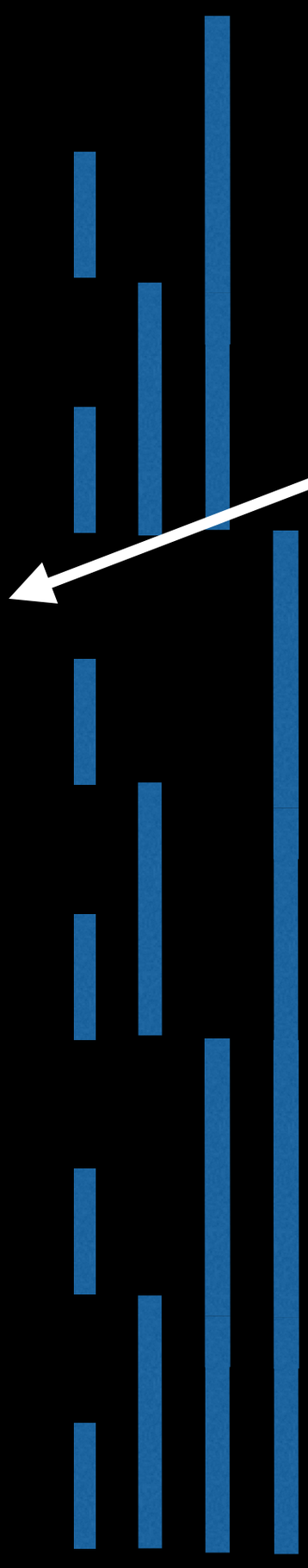
$i = 12 = 1100_2$

$j = 1100_2 + 0100_2 = 10000_2$
$= 16$

Ignore updating j if index is out of bounds

Linear Construction

12	1100_2	-11
11	1011_2	4
10	1010_2	-7
9	1001_2	-9
8	1000_2	23
7	0111_2	5
6	0110_2	14
5	0101_2	3
4	0100_2	12
3	0011_2	-2
2	0010_2	7
1	0001_2	3



Constructed Fenwick tree! We can now perform point and range query updates as required.

Fenwick Tree Source Code