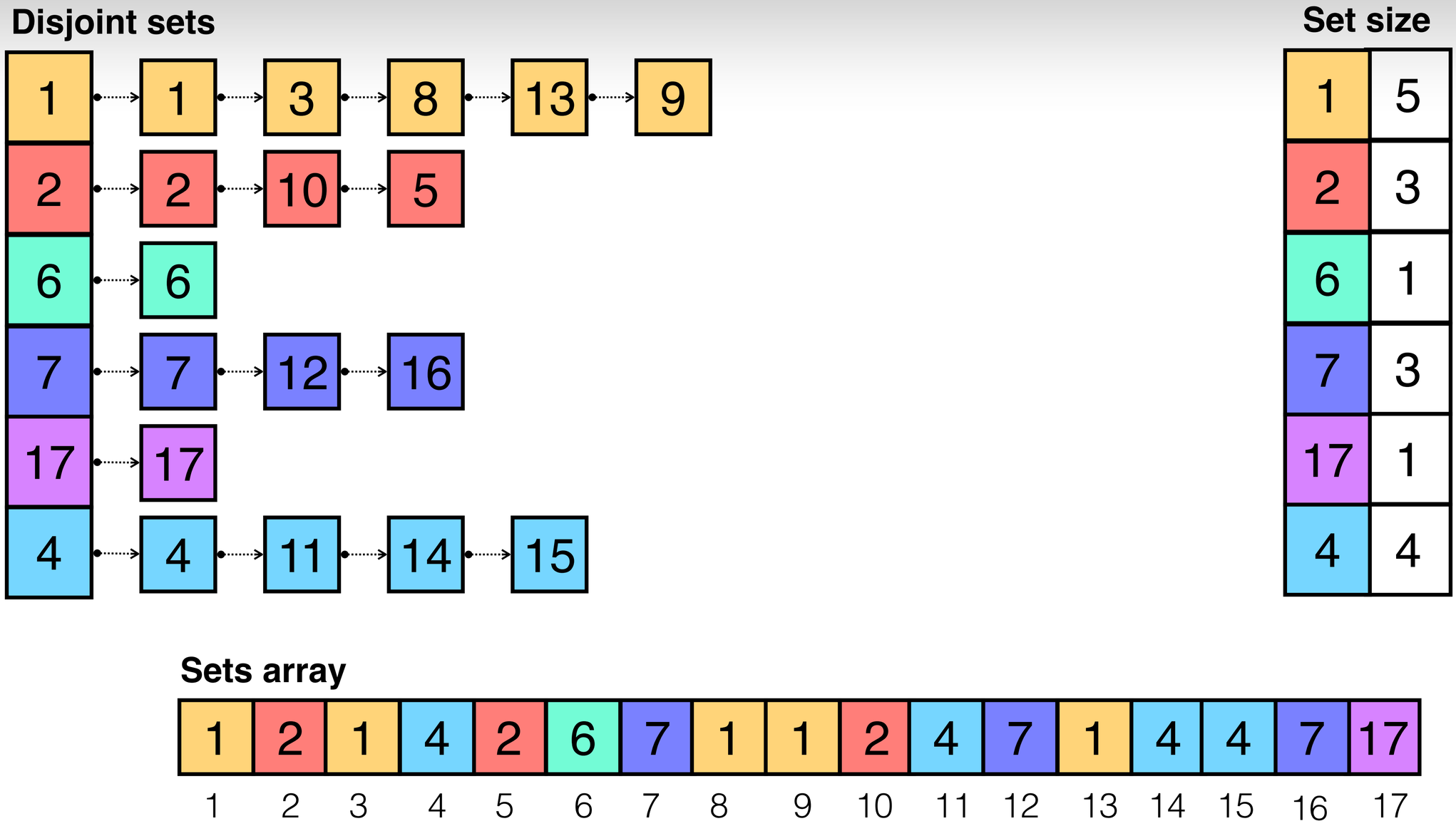
GRAPHS: UNION FIND

ISIS 2804

UNION FIND

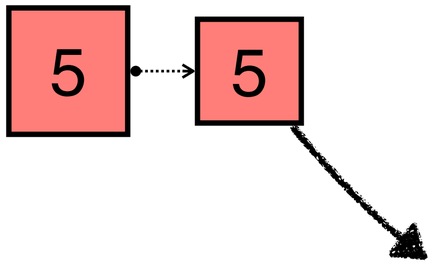
[Cormen et al. Chaper 21]

Union-find


Disjoint sets
1 1 3 8 13 9
2 2 10 5
6 6
7 7 12 16
17 17
4 4 11 14 15
Set size
1 5
2 3
6 1
7 3
17 1
4 4
Sets array
1 2 1 4 2 6 7 1 1 2 4 7 1 4 4 7 17
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

make

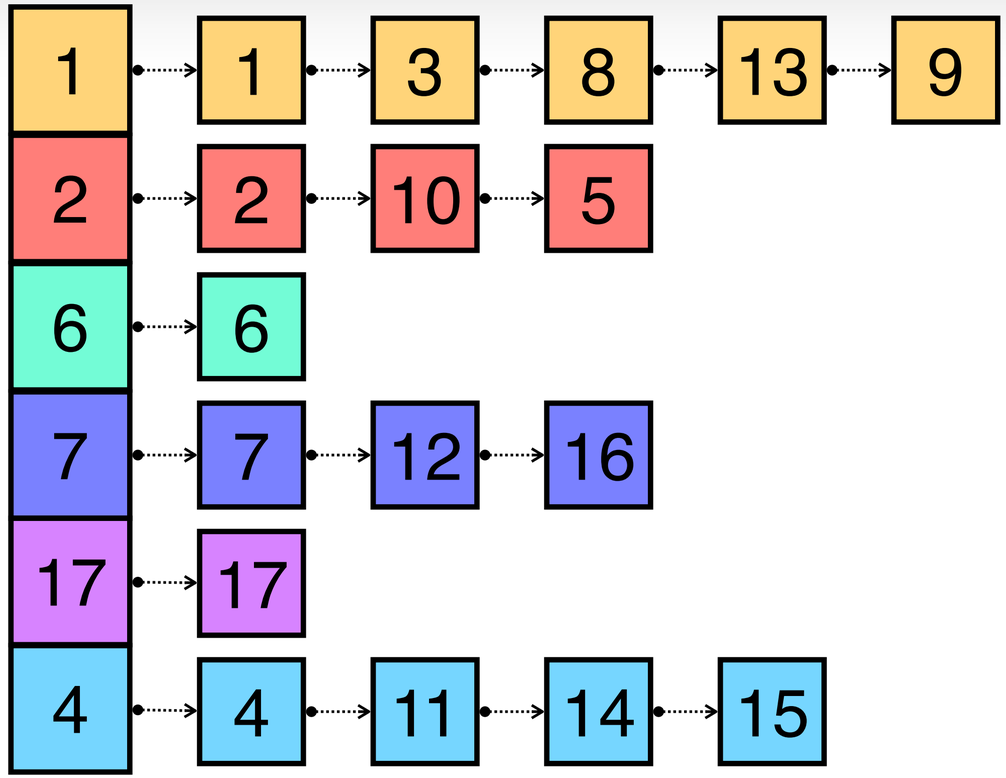
```
make(5)
```



Create a new disjoint set with the given elements and get a representative

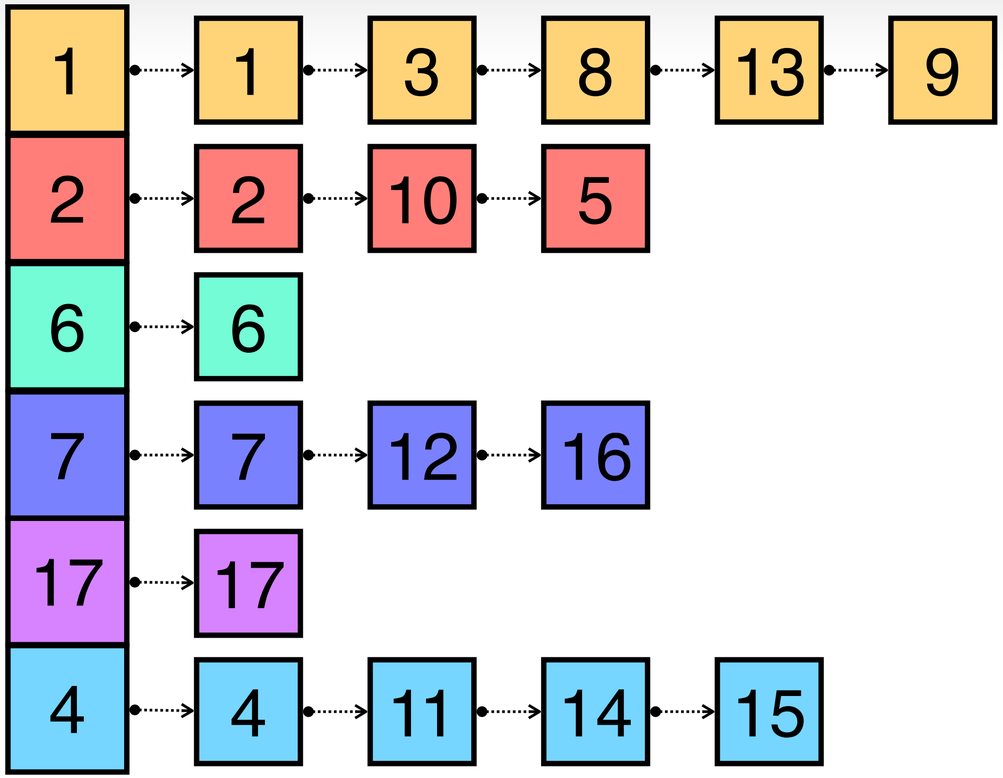
find

`find(5)`



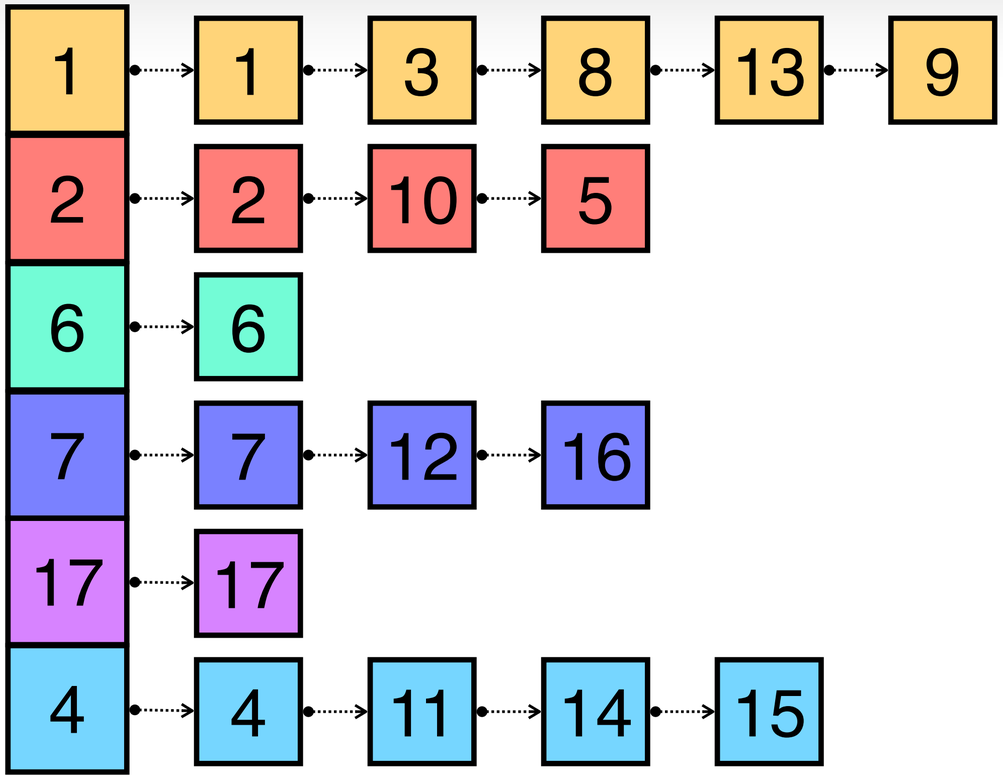
find

```
find(5)
=
```



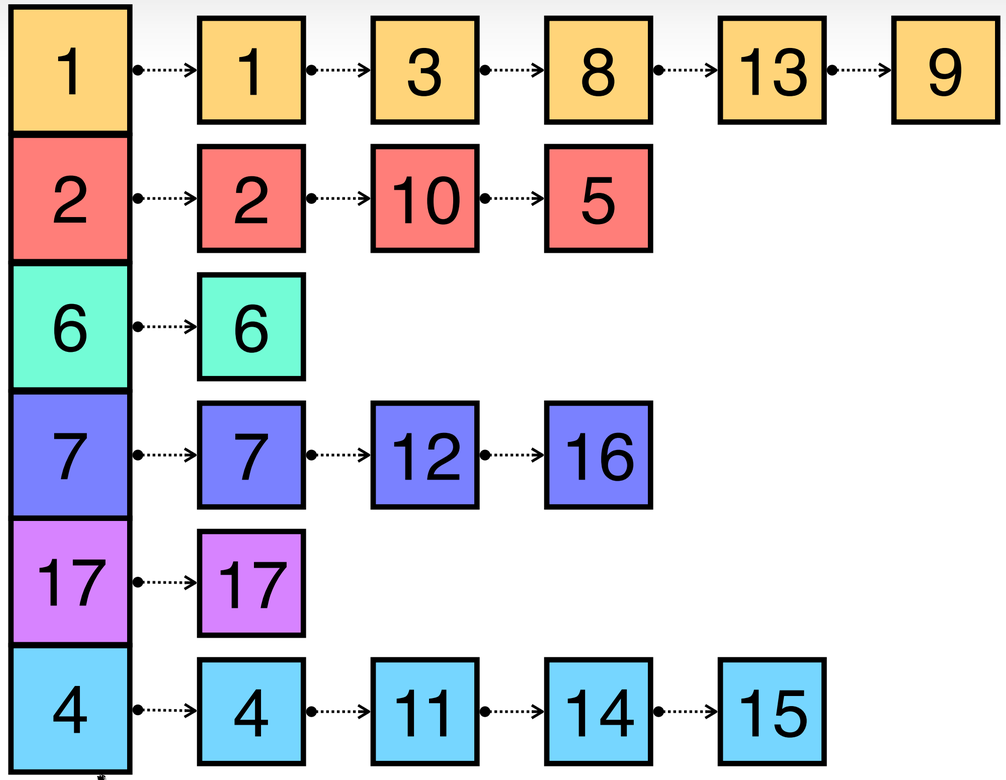
find

```
find(5)

=

2
```



find

```
find(5)
=
2
```

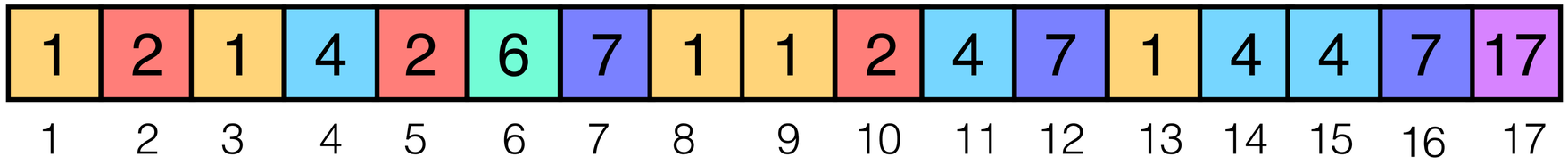


look for the element among **all** elements in **every** set

WHATS THE COMPLEXITY OF FIND?

find

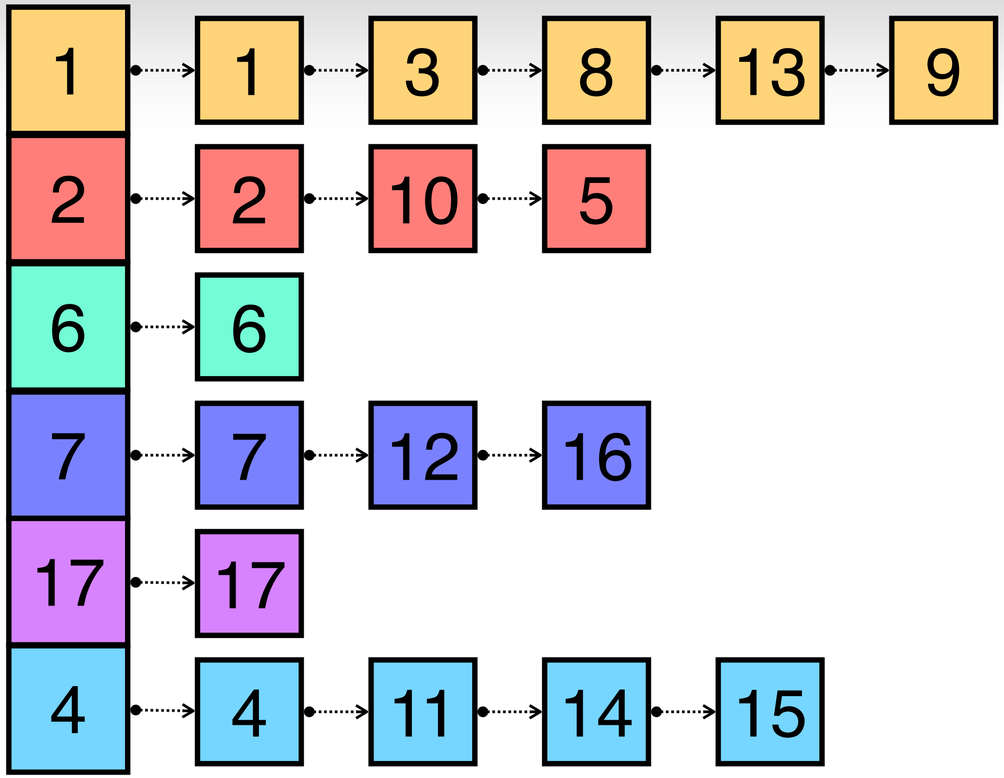
```
find(5)
```

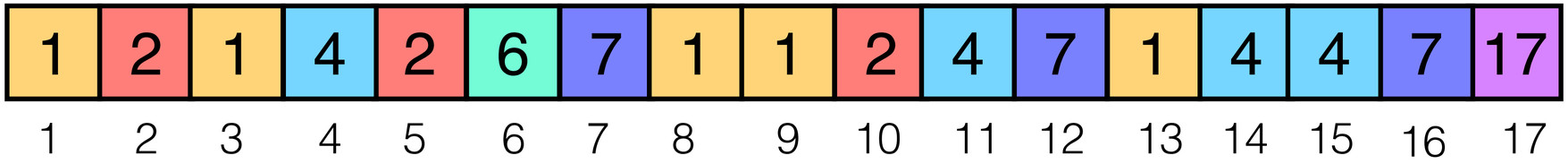


O(N)

find

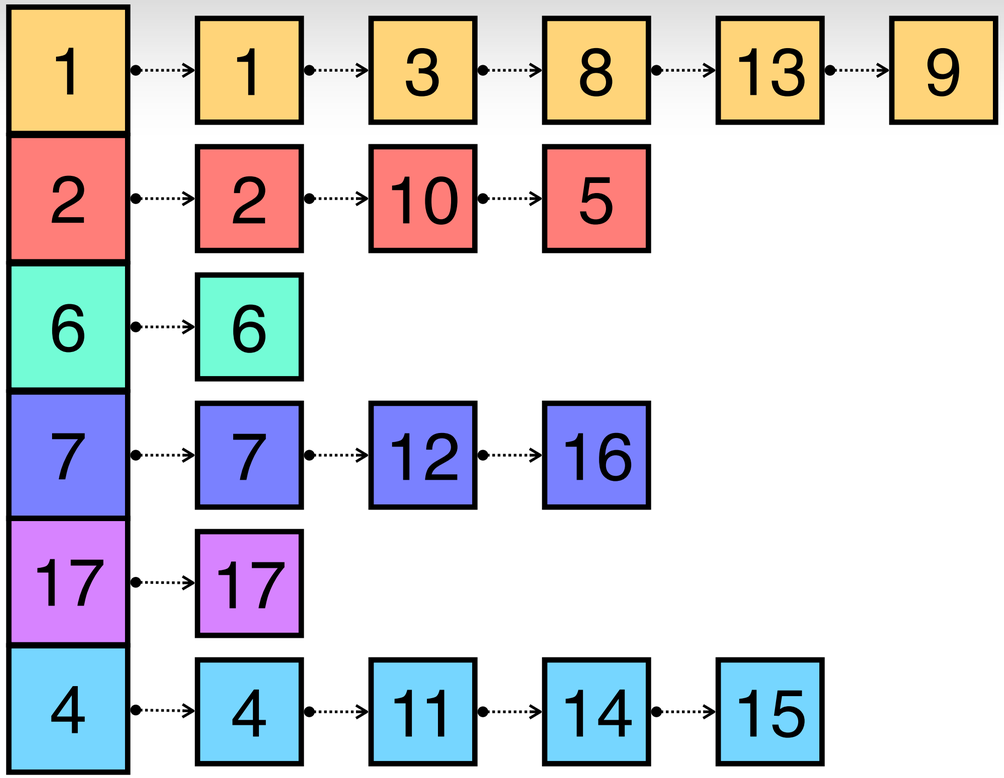
`find(5)`


1
1
3
8
13
9
2
2
10
5
6
6
7
7
12
16
17
17
4
4
11
14
15

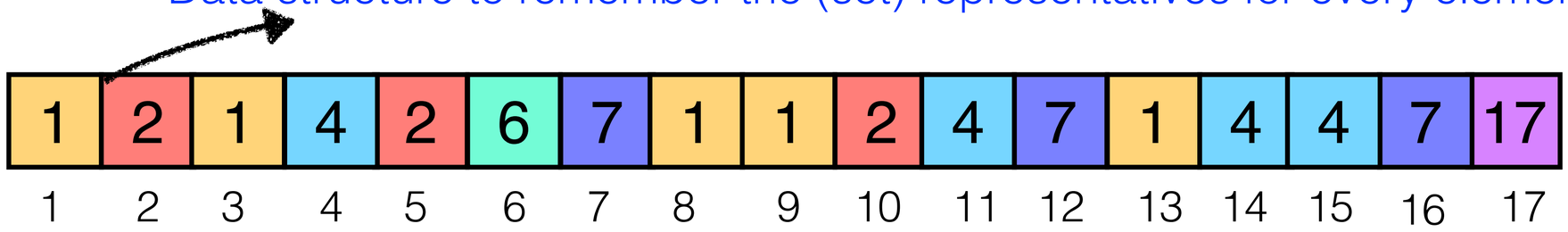

1 2 1 4 2 6 7 1 1 2 4 7 1 4 4 7 17
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

find

`find(5)`



Data structure to remember the (set) representatives for every element

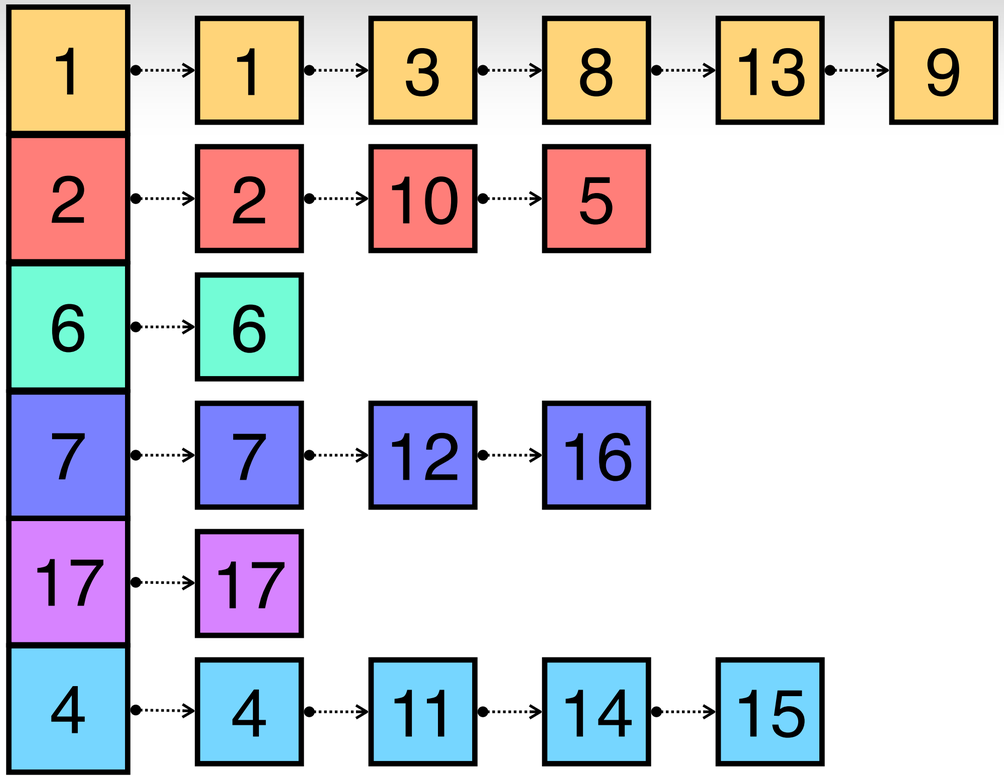


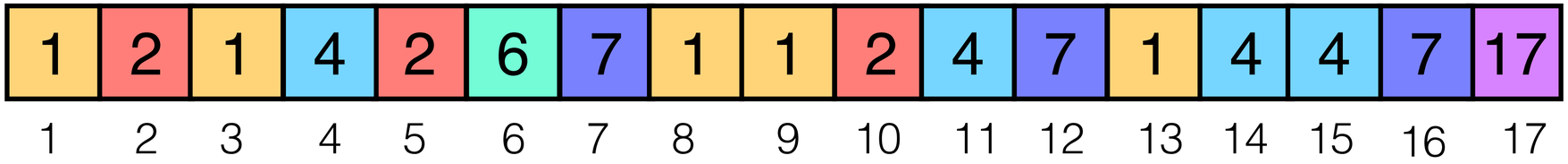
COMPLEXITY?

find

```
find(5)
```

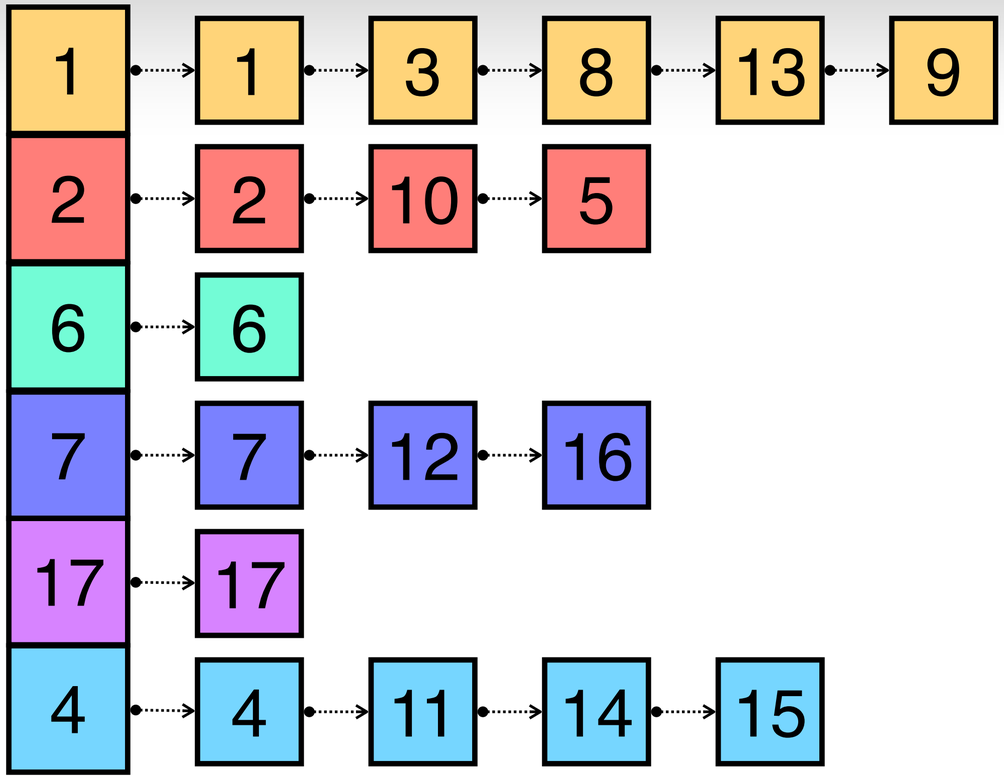
O(1)





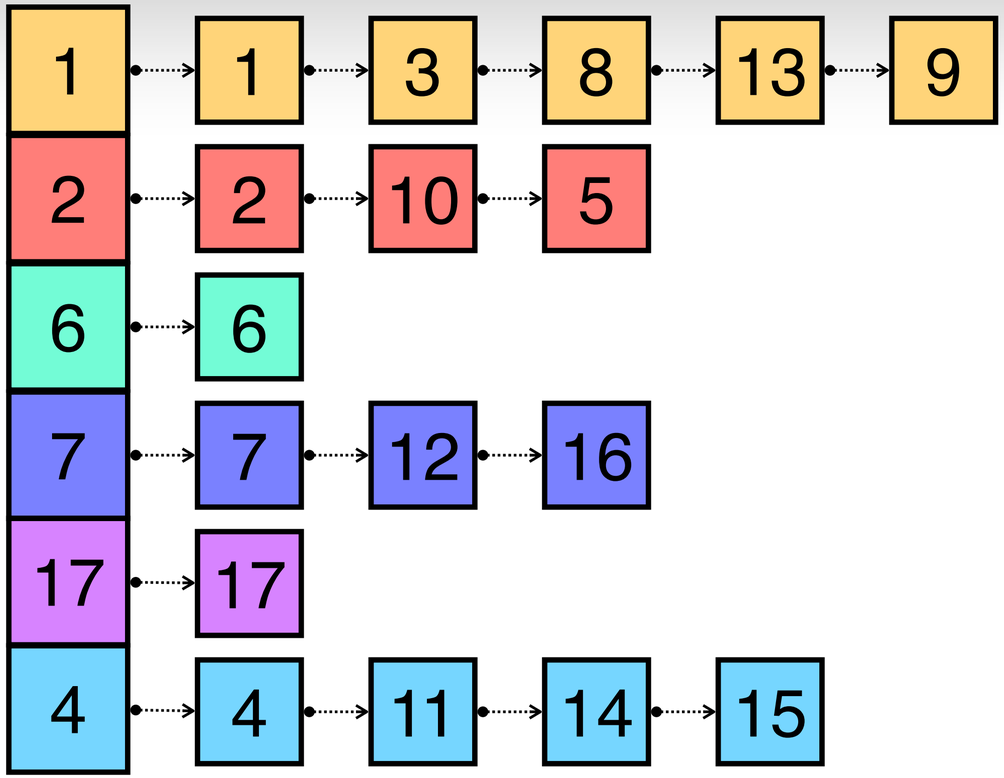
count

`count(13)`



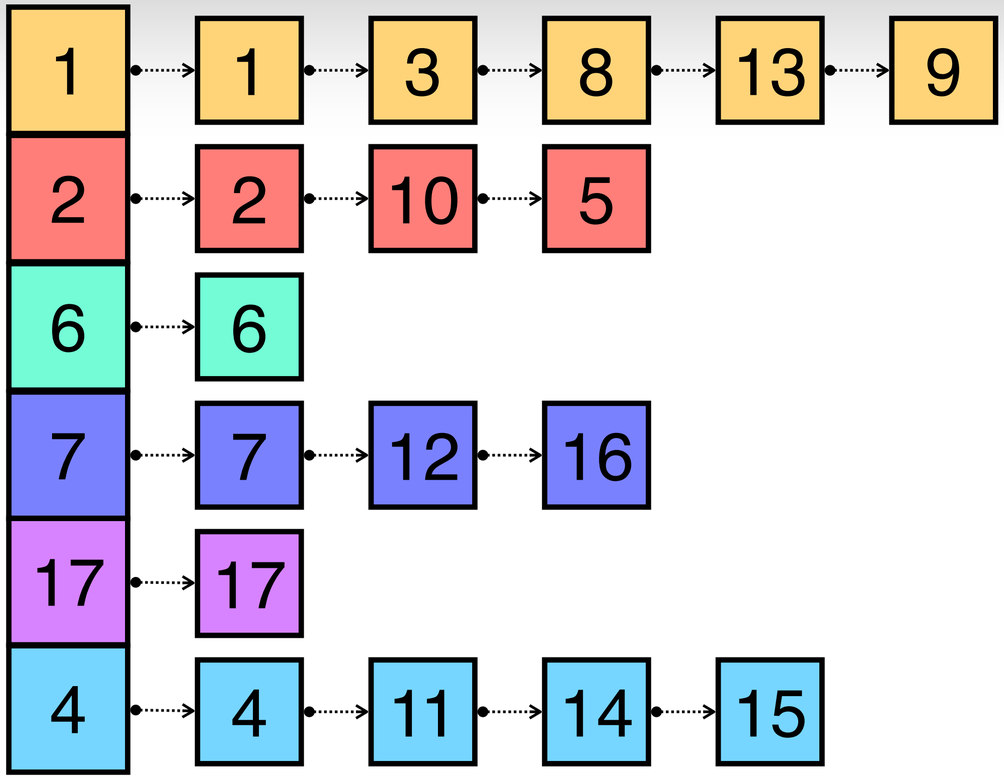
count

```
count(13)

=
```



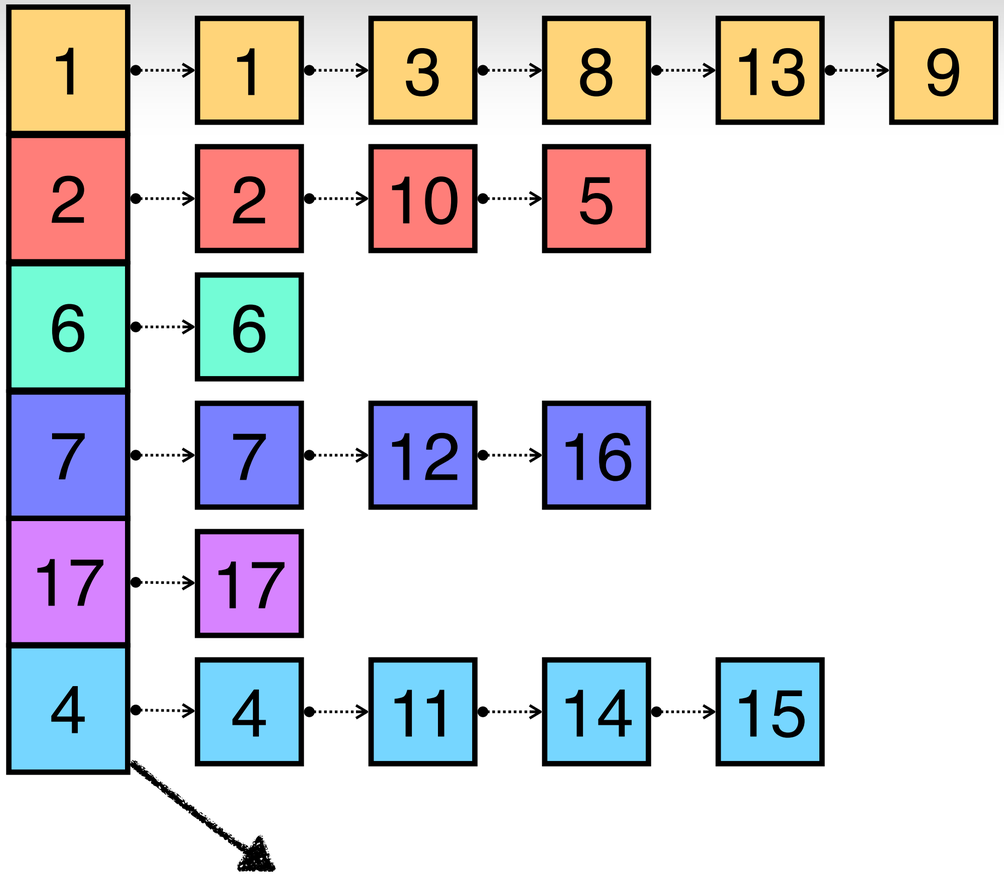
count

```
count(13)

=

5
```



count

```
count(13)

=

5
```

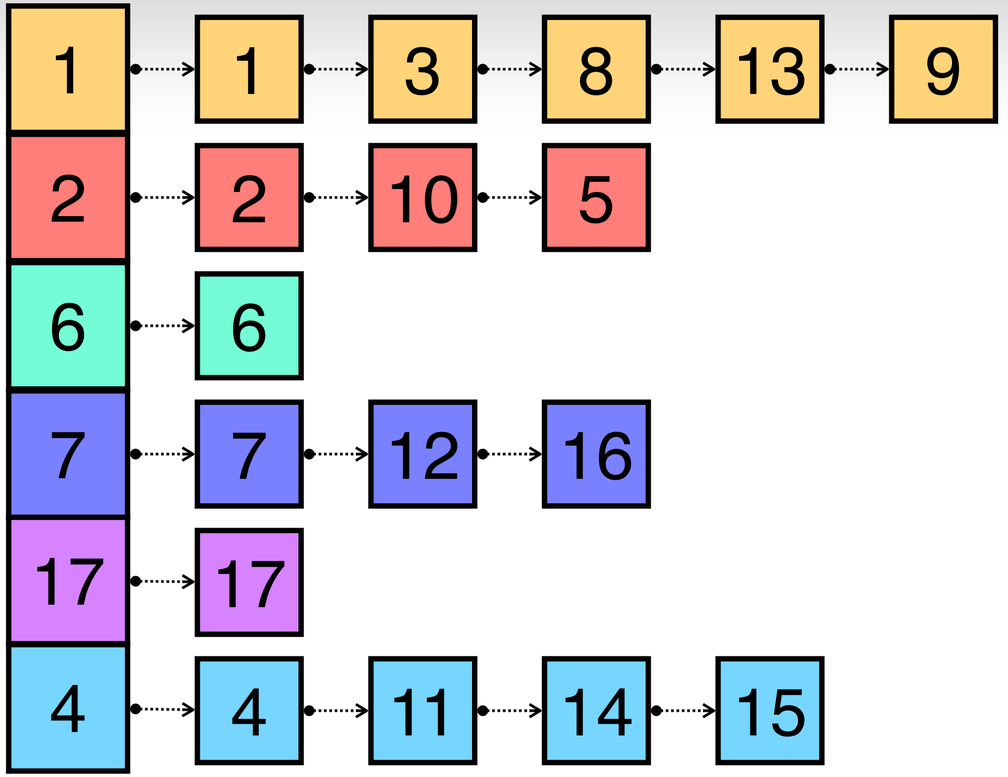


Must search the element **between all** the elements of **every representative** and count the elements of such representative

COMPLEXITY OF COUNT

count

`count(13)`

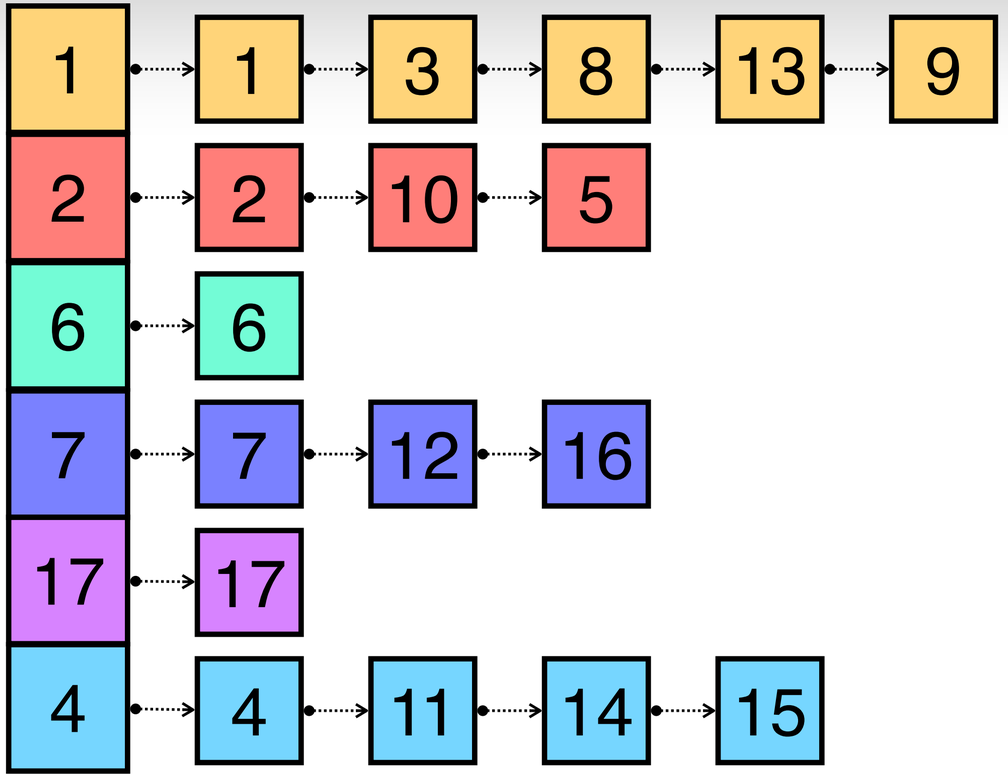


1	5
2	3
6	1
7	3
17	1
4	4

count

`count(13)`

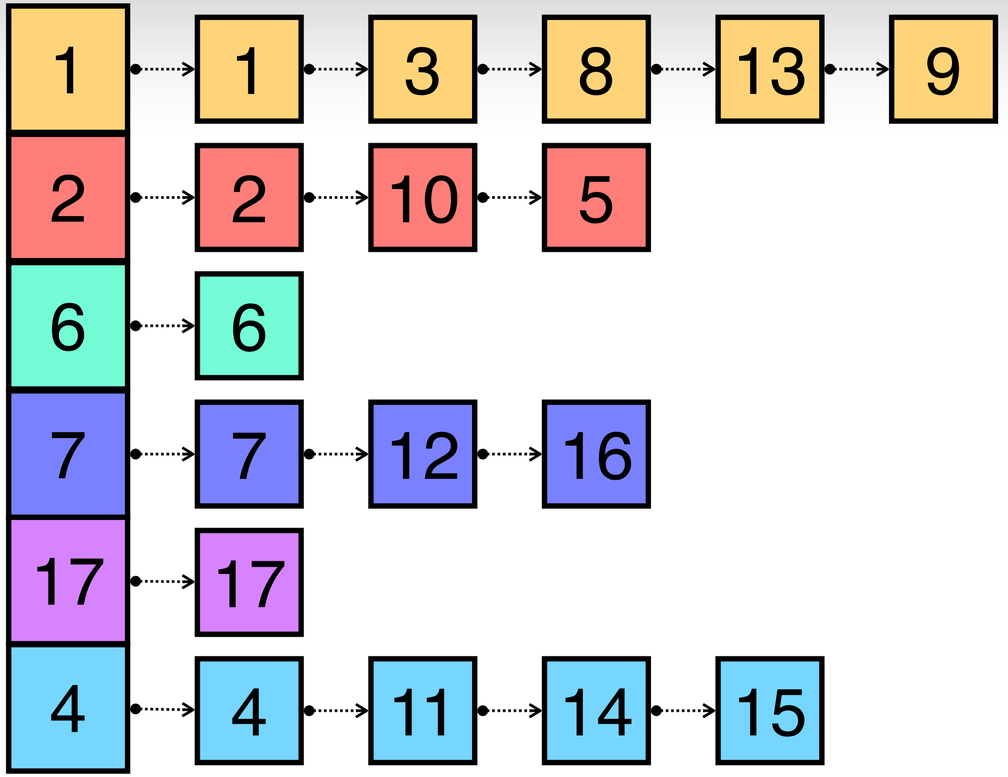
=



1	5
2	3
6	1
7	3
17	1
4	4

count

```
count(13)

=

5
```



1	5
2	3
6	1
7	3
17	1
4	4

count

```
public int count(int x) {
 int rep = find(x);
  for(int i=0; i<counts.length; i++) {
   if(counts[i] == rep)
     return counts[i].value;
  }
}
```

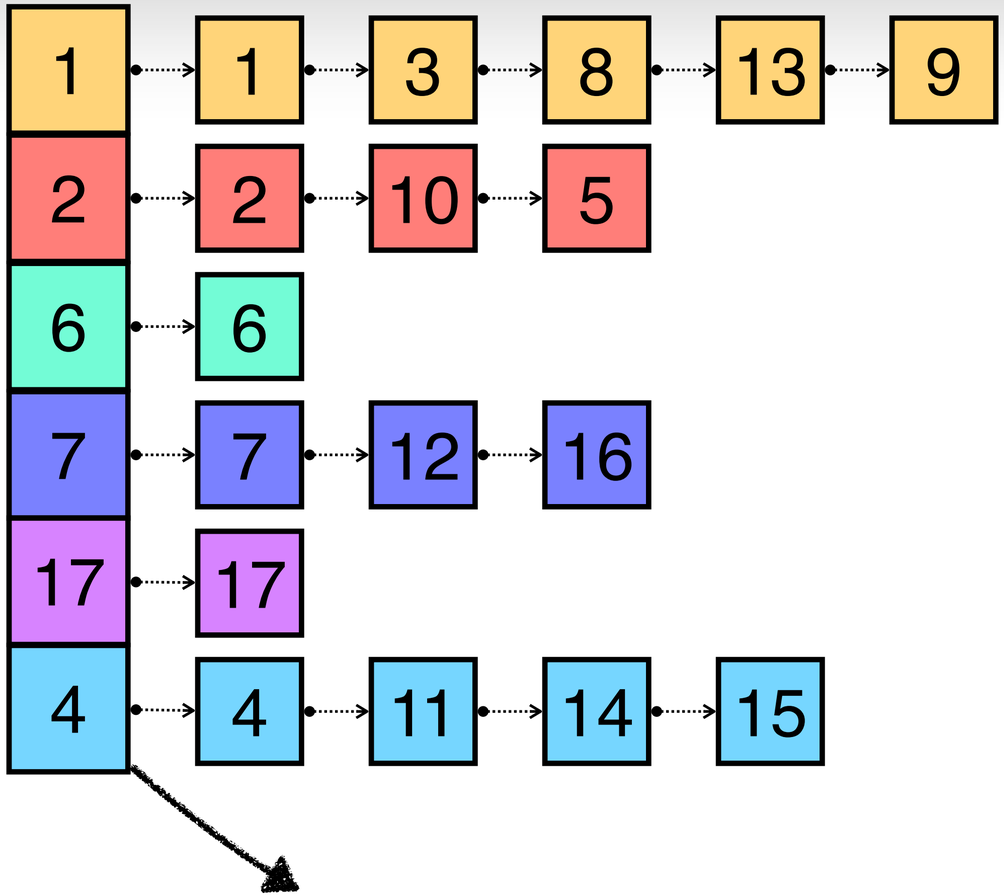
count

```
public int count(int x) {
 int rep = find(x);
  for(int i=0; i<counts.length; i++) {
   if(counts[i] == rep)
     return counts[i].value;
  }
}
```

COMPLEXITY = O(|S|)

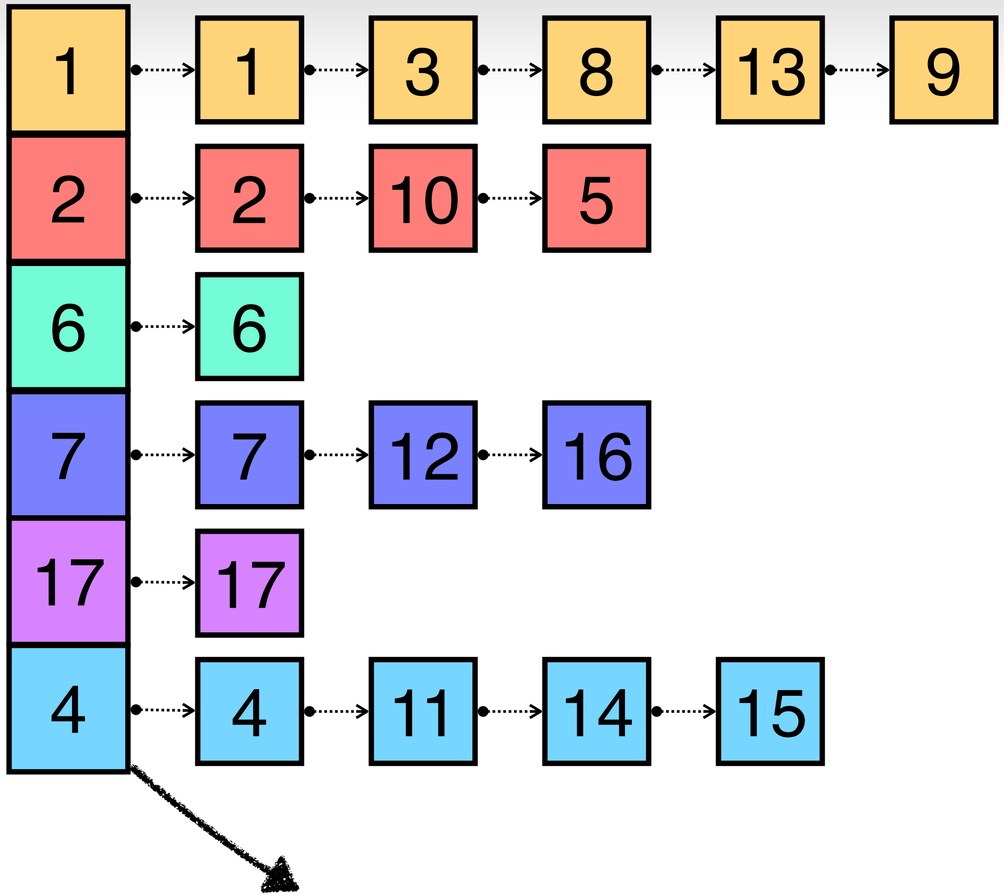
connected

```
connected(13, 3)
```



connected

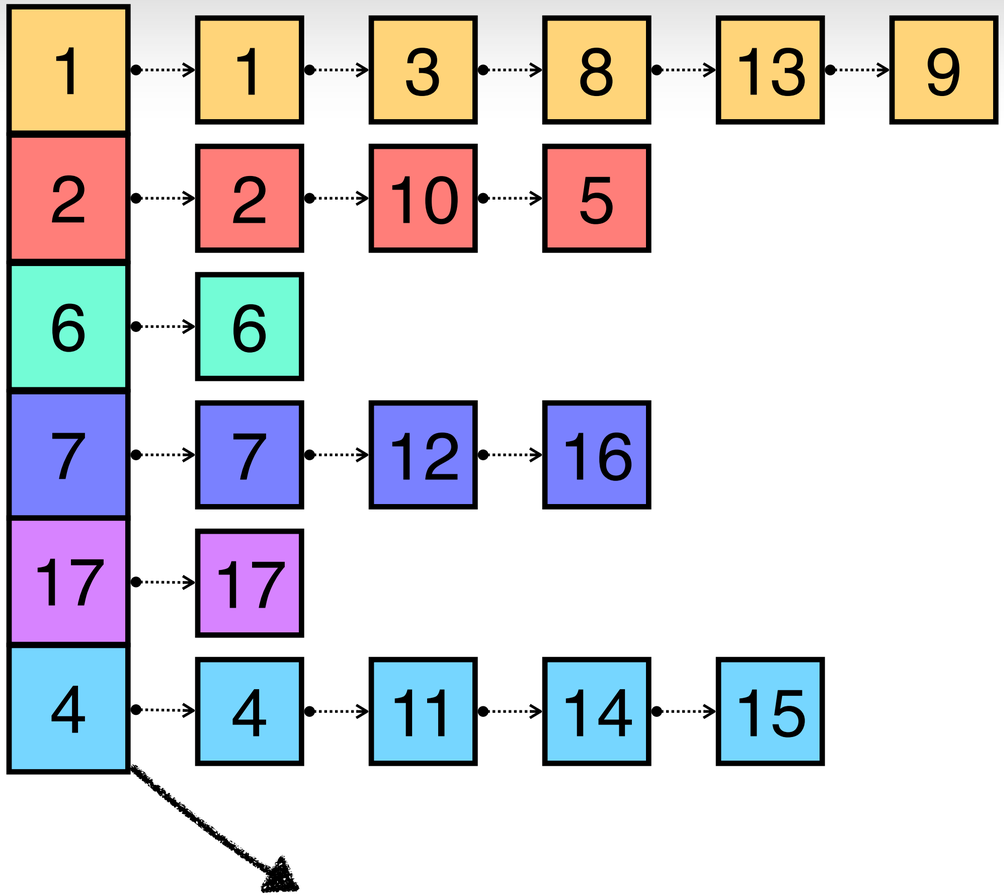
```
connected(13, 3)
```



Must search the elements **among all** the elements in **every representative**

connected

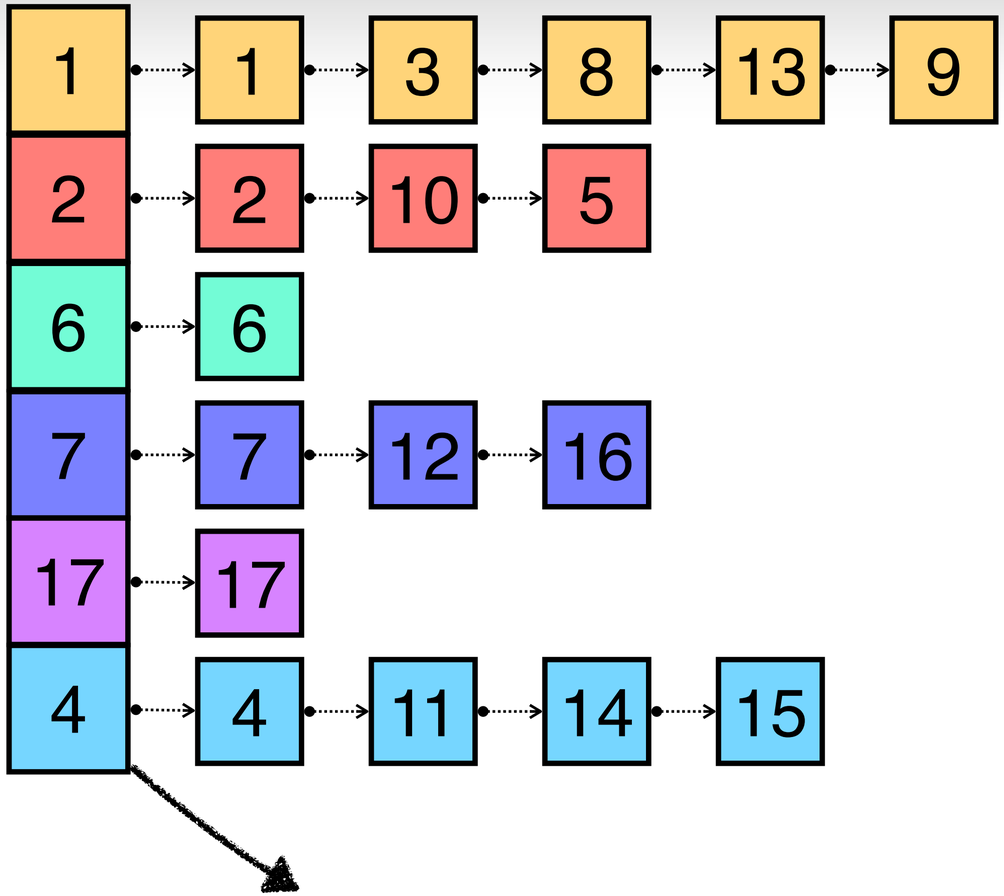
```
connected(13, 3)
=
```



Must search the elements **among all** the elements in **every representative**

connected

```
connected(13, 3)
=
true
```



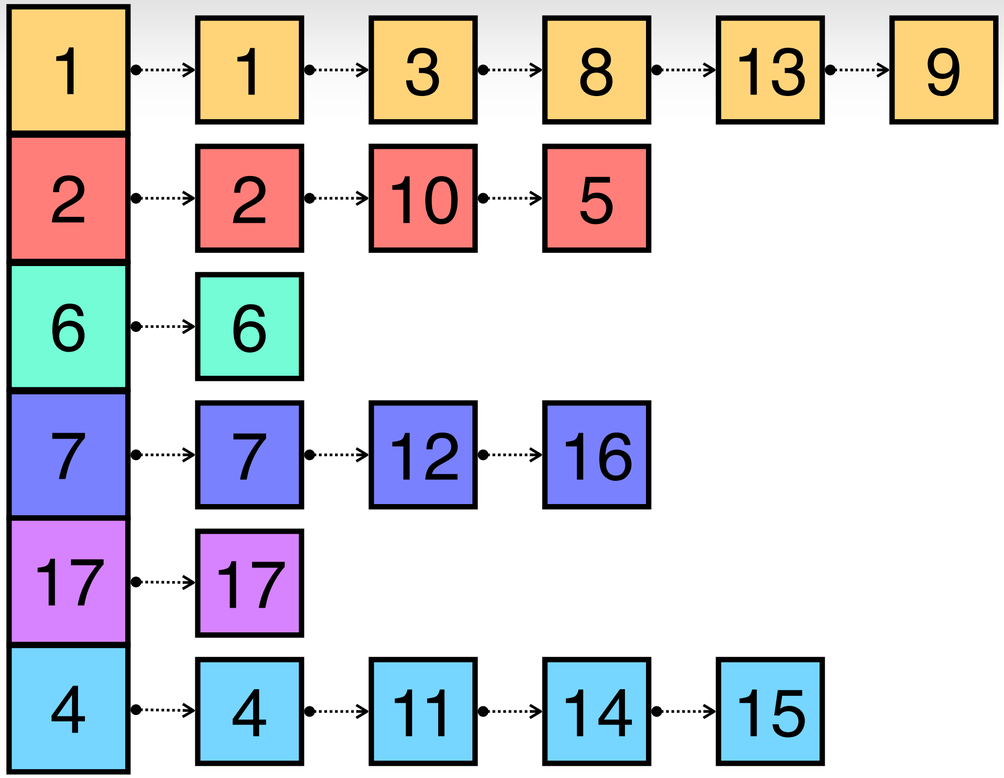
Must search the elements **among all** the elements in **every representative**

COMPLEXITY OF CONNECTED

connected

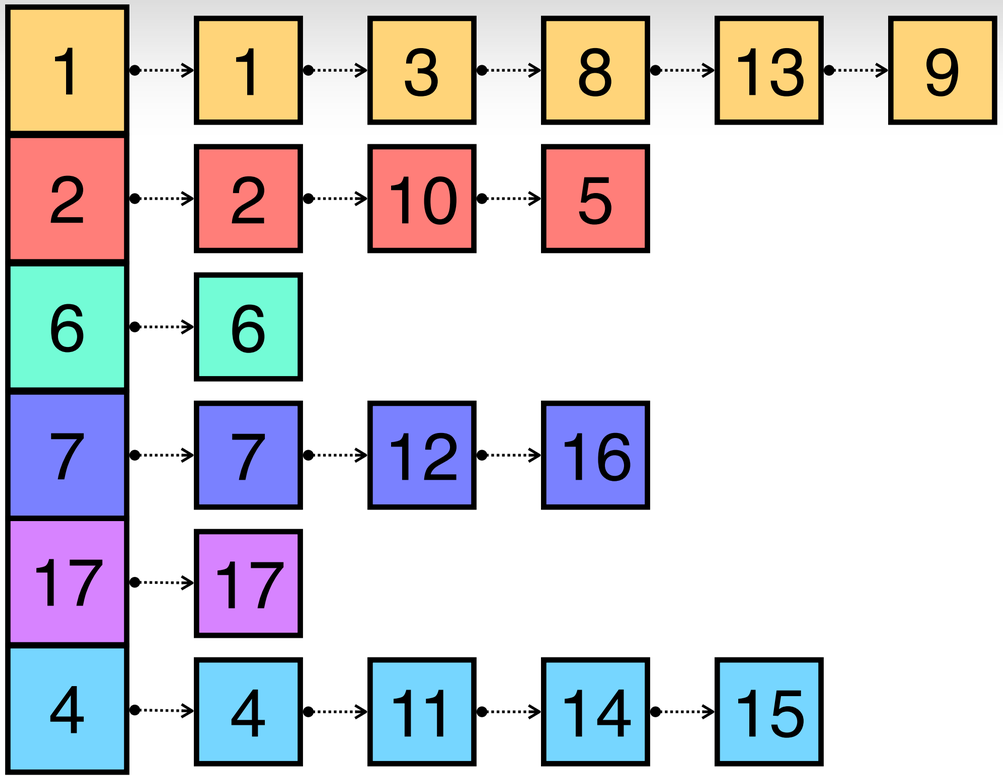
```
connected(13, 3)
```

O(N)



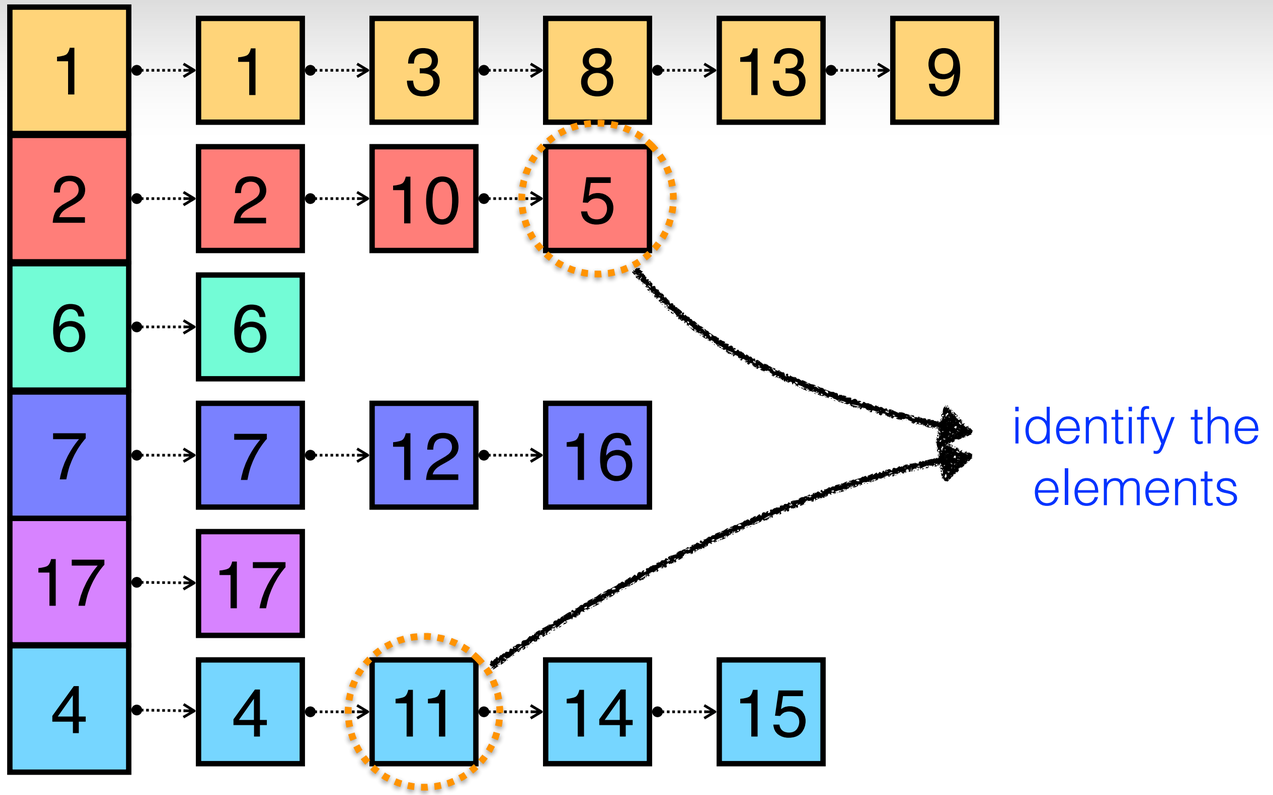
union

```
union(5, 11)
```



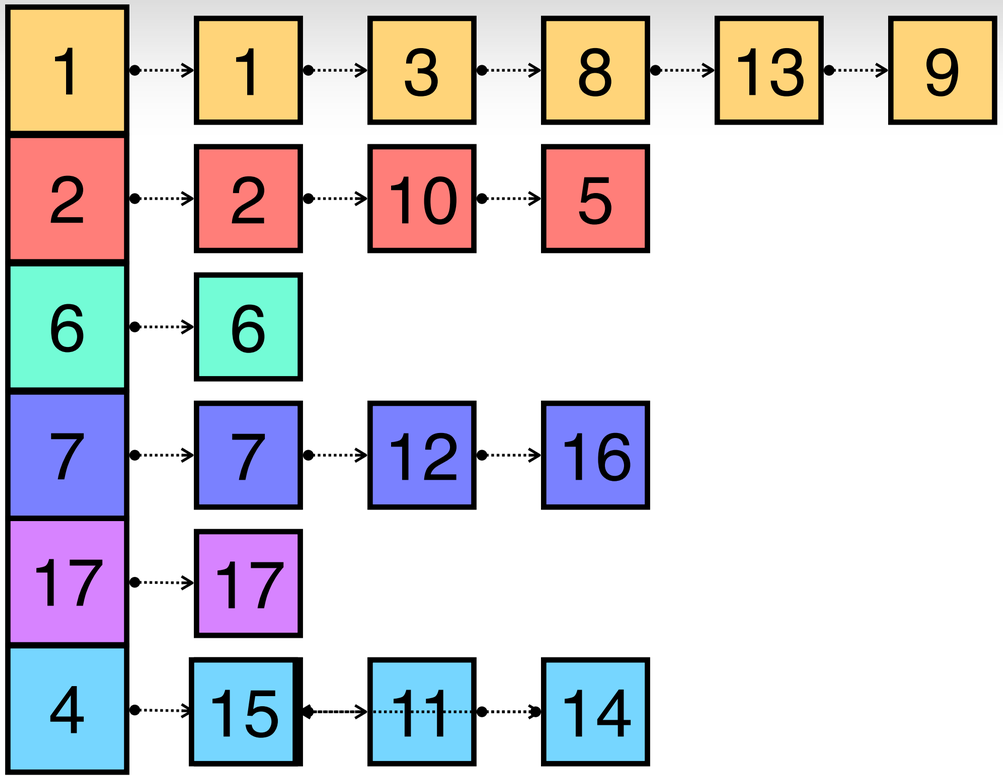
union

`union(5, 11)`



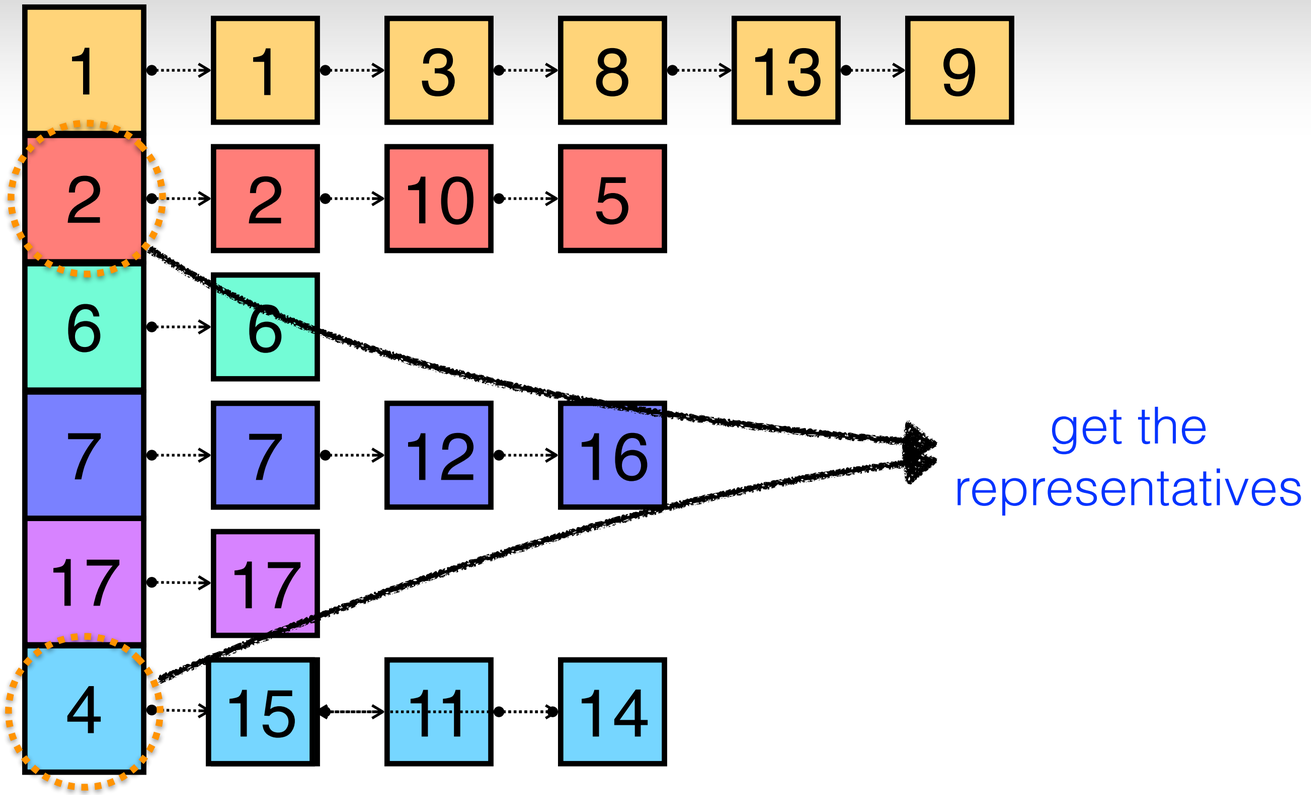
union

```
union(5, 11)
```



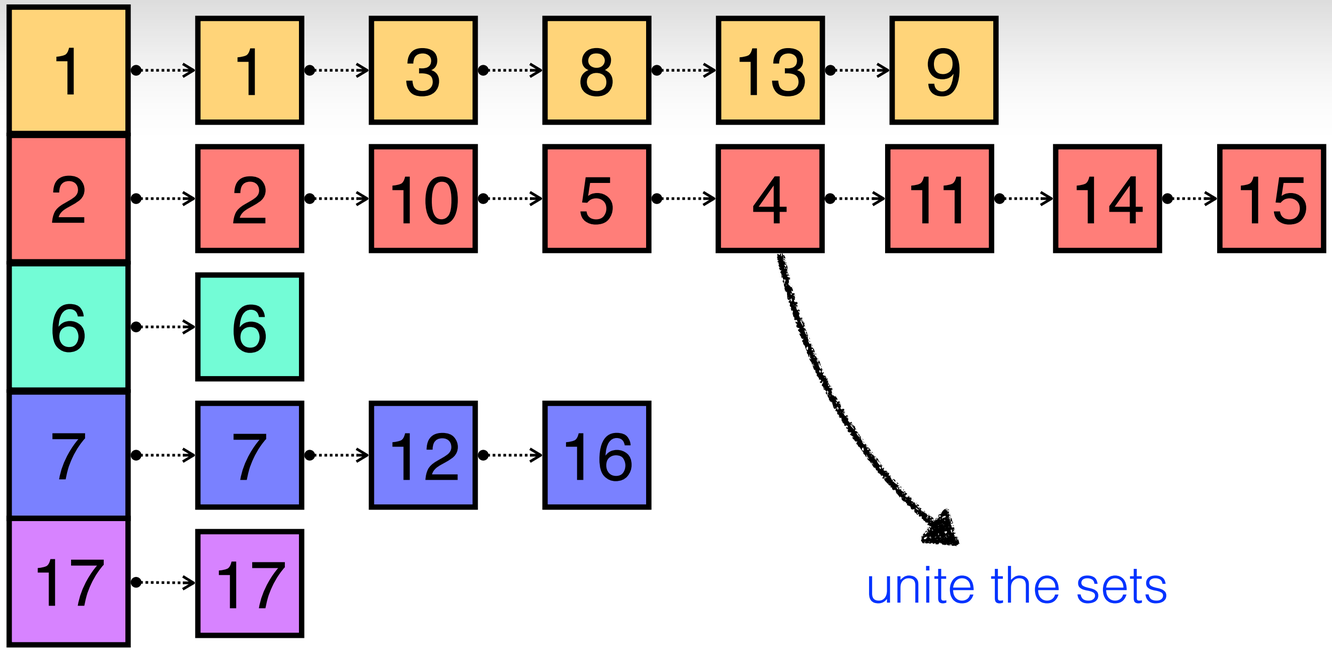
union

`union(5, 11)`



union

`union(5, 11)`



COMPLEXITY OF UNION

Union

```
public void union(int x, int y) {
 int rep1 = find(x);
 int rep2 = find(y);
 if(count(x) < count(y)) {
  sets.get(rep2).value.concat(sets.ge
  t(rep1).value);
 else
  sets.get(rep1).value.concat(sets.ge
  t(rep2).value);
}
```

Union

If you do `k` unions, maximum `2k` elements belong to such unions.

Every time a set changes, its size **at least** doubles (over approximation).

Therefore, a set can change a maximum of `log(2k)` times

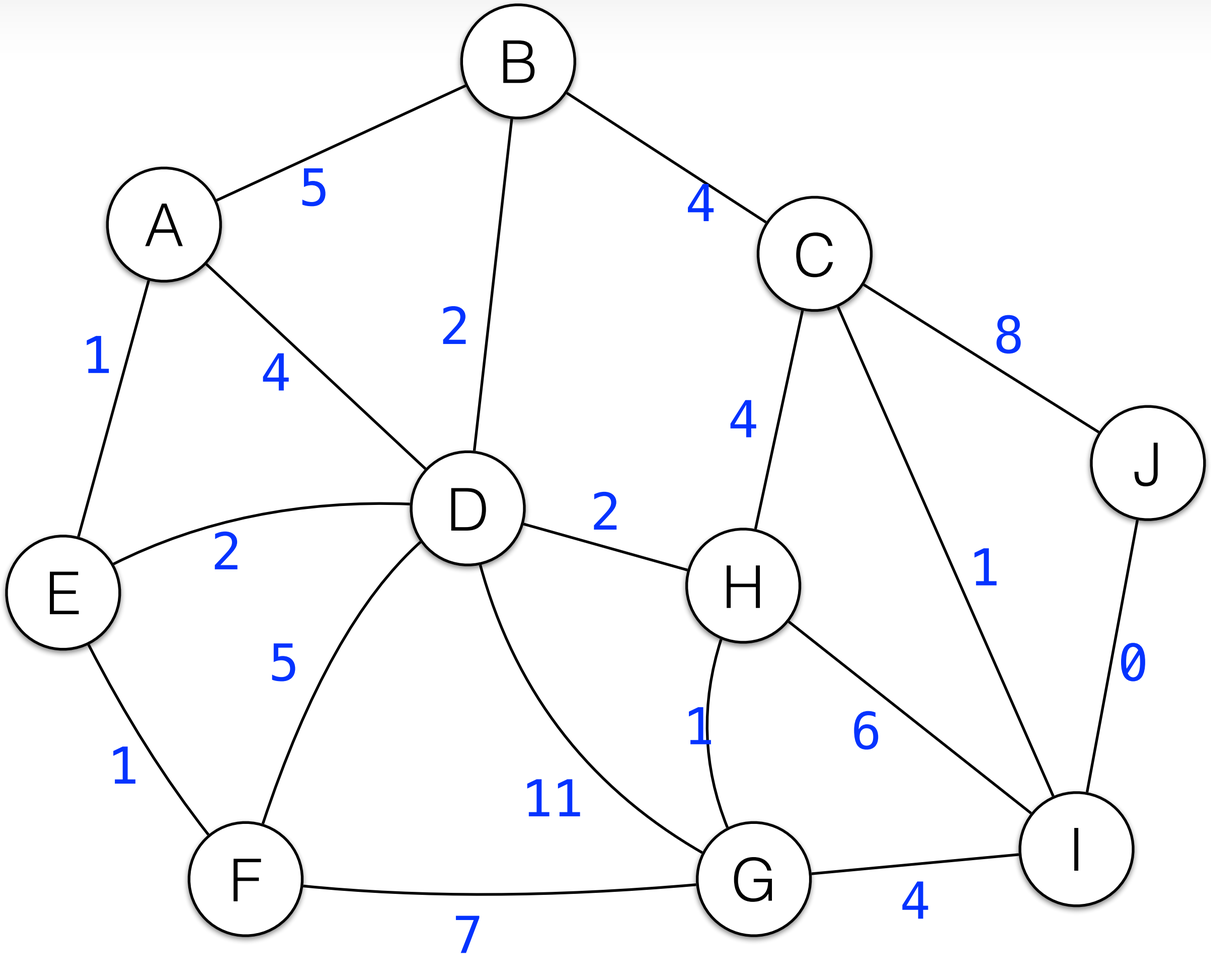
Changing `2k` elements, each is updated maximum `log(2k)` times, so the complexity is **`2klog(2k)`**.

Path compression

```
vector<int> parent(n);
void make_set(int v) {
    parent[v] = v;
}

int find_set(int v) {
    if (v == parent[v])
        return v;
    return parent[v] = find_set(parent[v]);
}

void union_sets(int a, int b) {
    a = find_set(a);
    b = find_set(b);
    if (a != b)
        parent[b] = a;
}
```

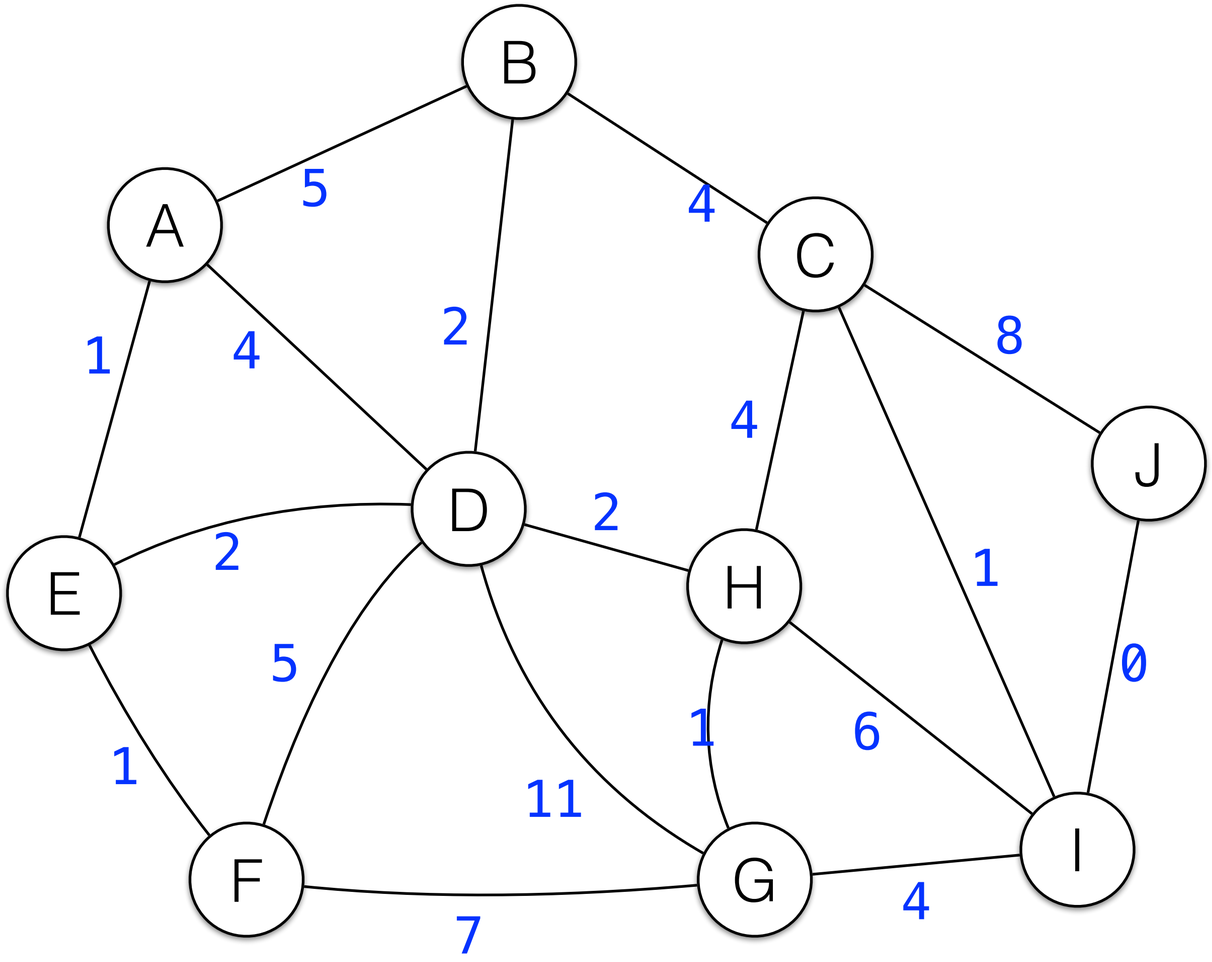
O(log n)

MST



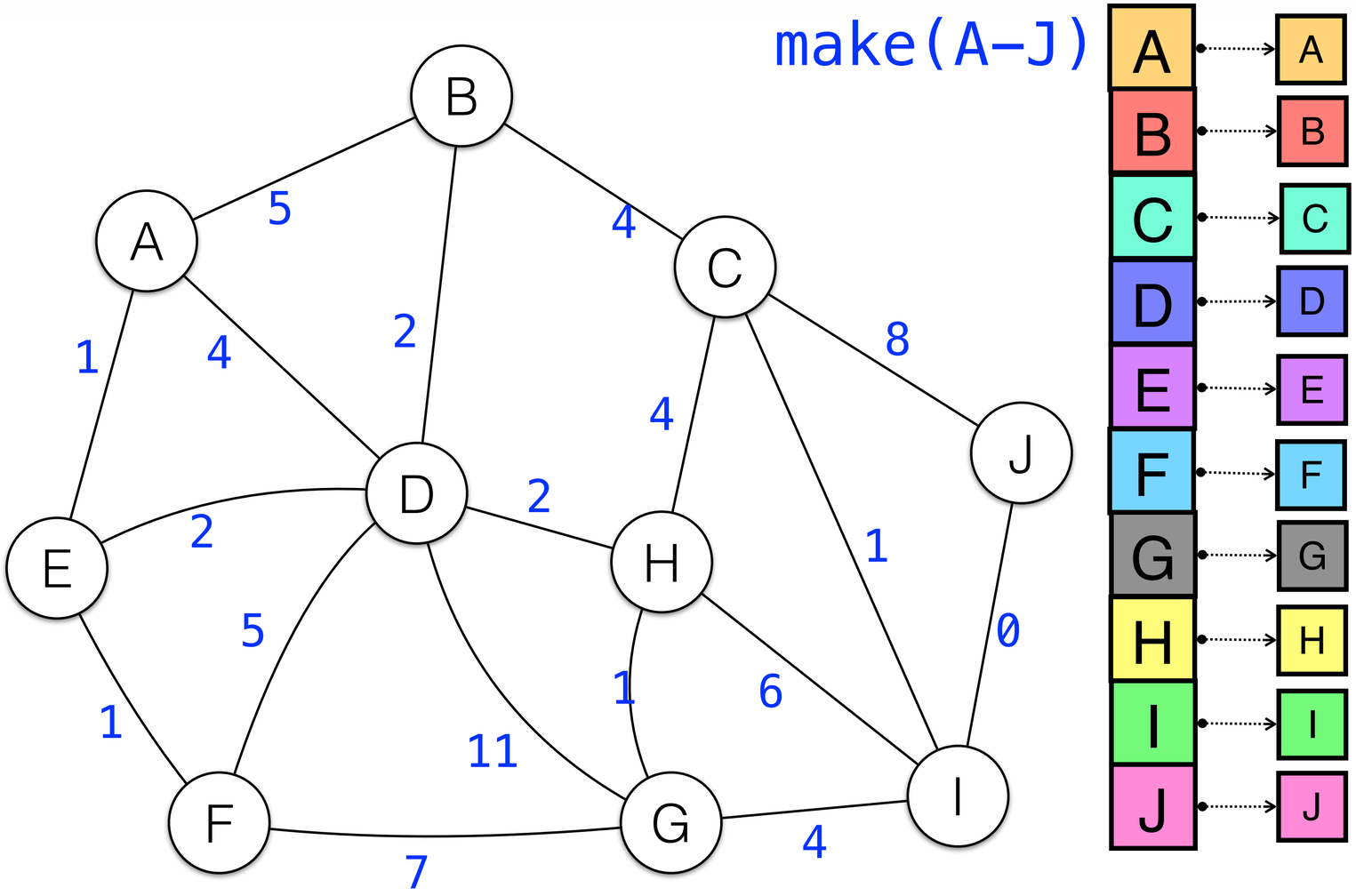
Kruskal MST

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



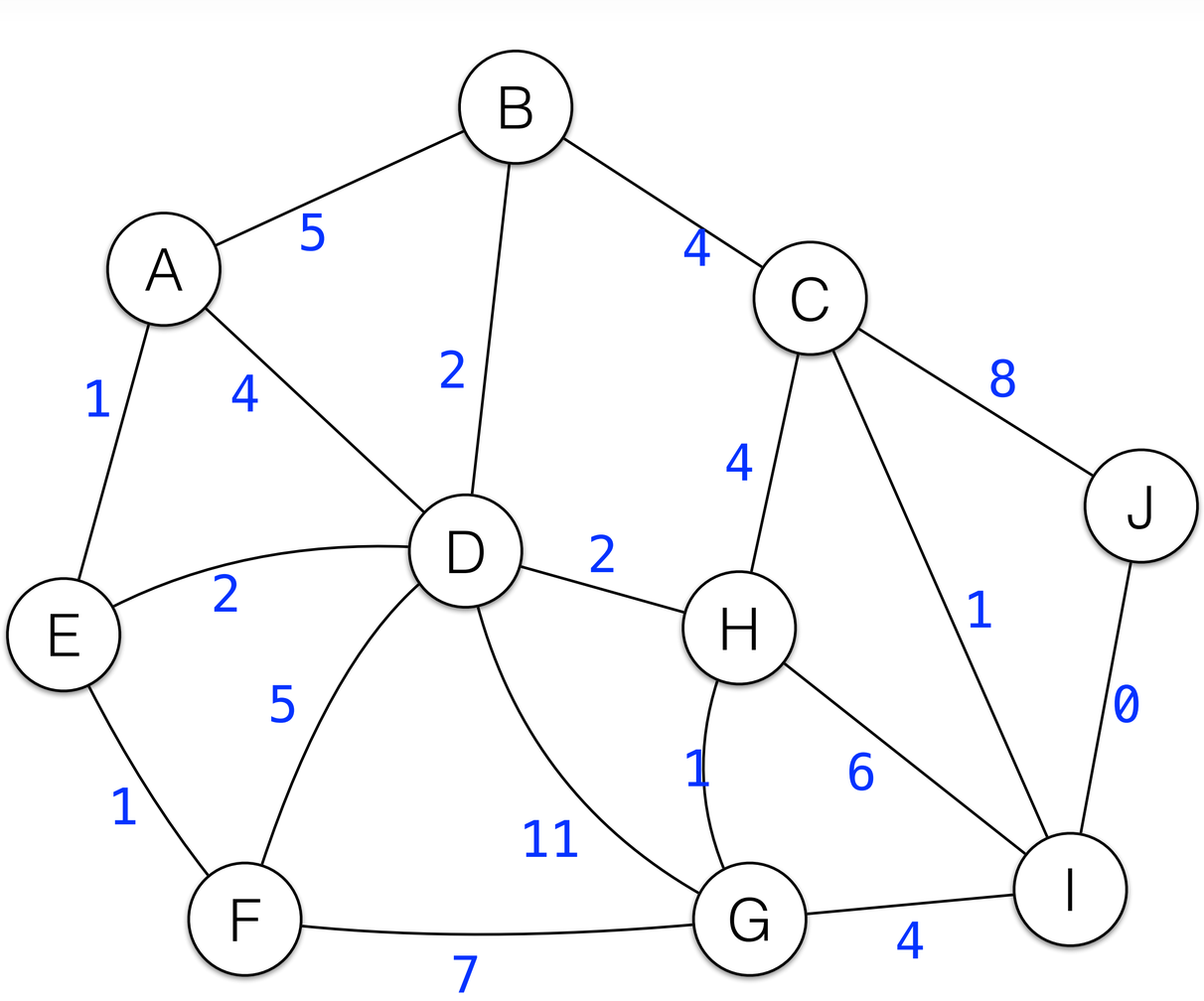
Kruskal MST

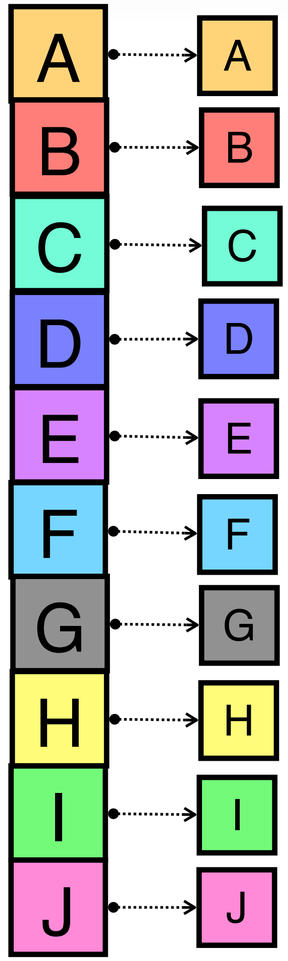
	I-J	A-E	C-I	E-F	G-H	B-D	C-J	D-E	D-H	A-D	B-C	C-H	G-I	A-B	D-F	H-I	F-G	D-G
d	0	1	1	1	1	2	2	2	2	4	4	4	4	5	5	6	7	11



Kruskal MST

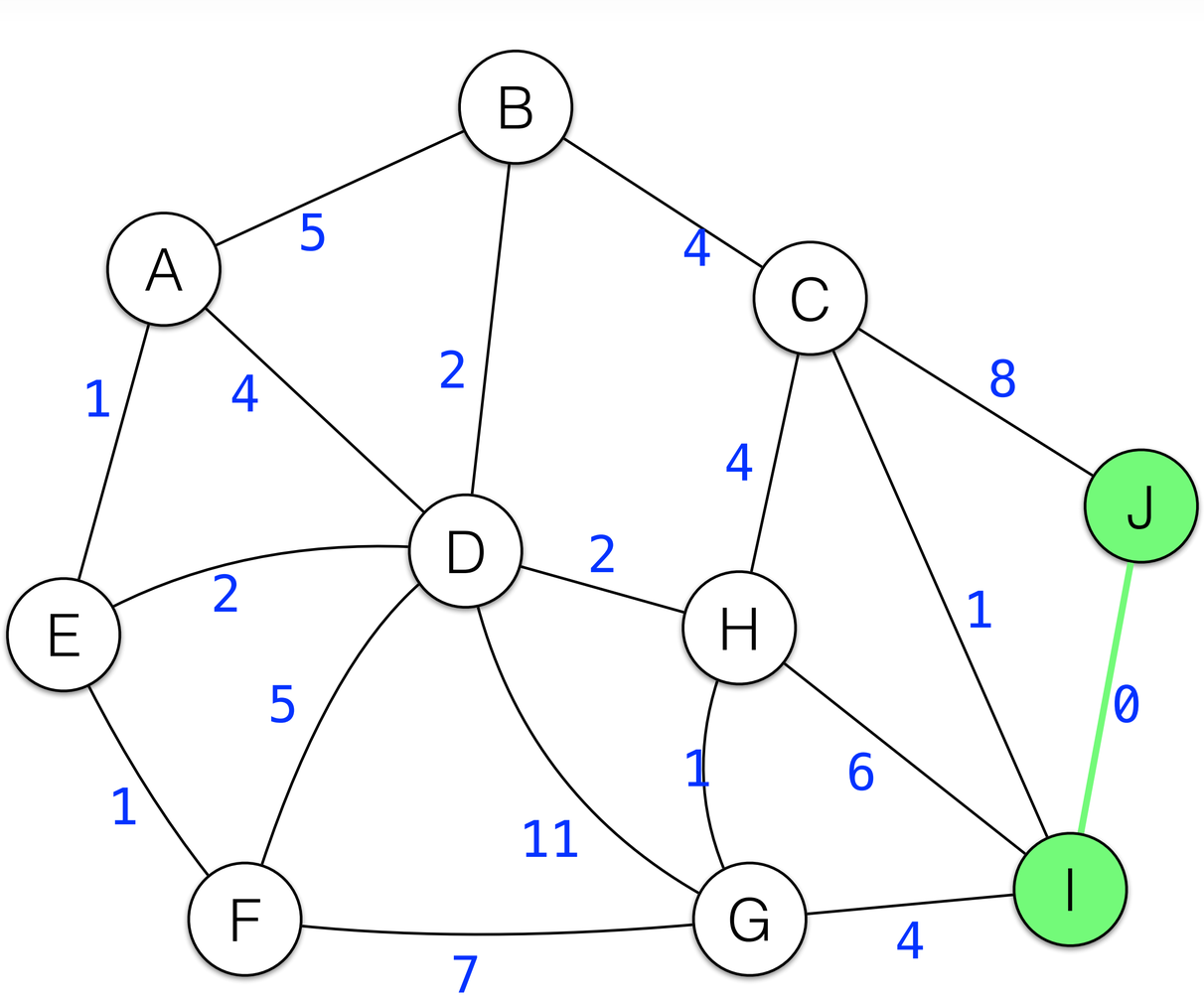
	I-J	A-E	C-I	E-F	G-H	B-D	C-J	D-E	D-H	A-D	B-C	C-H	G-I	A-B	D-F	H-I	F-G	D-G
d	0	1	1	1	1	2	2	2	2	4	4	4	4	5	5	6	7	11

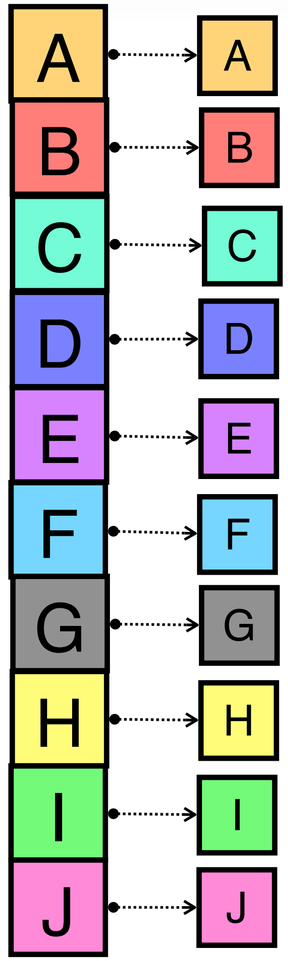




Kruskal MST

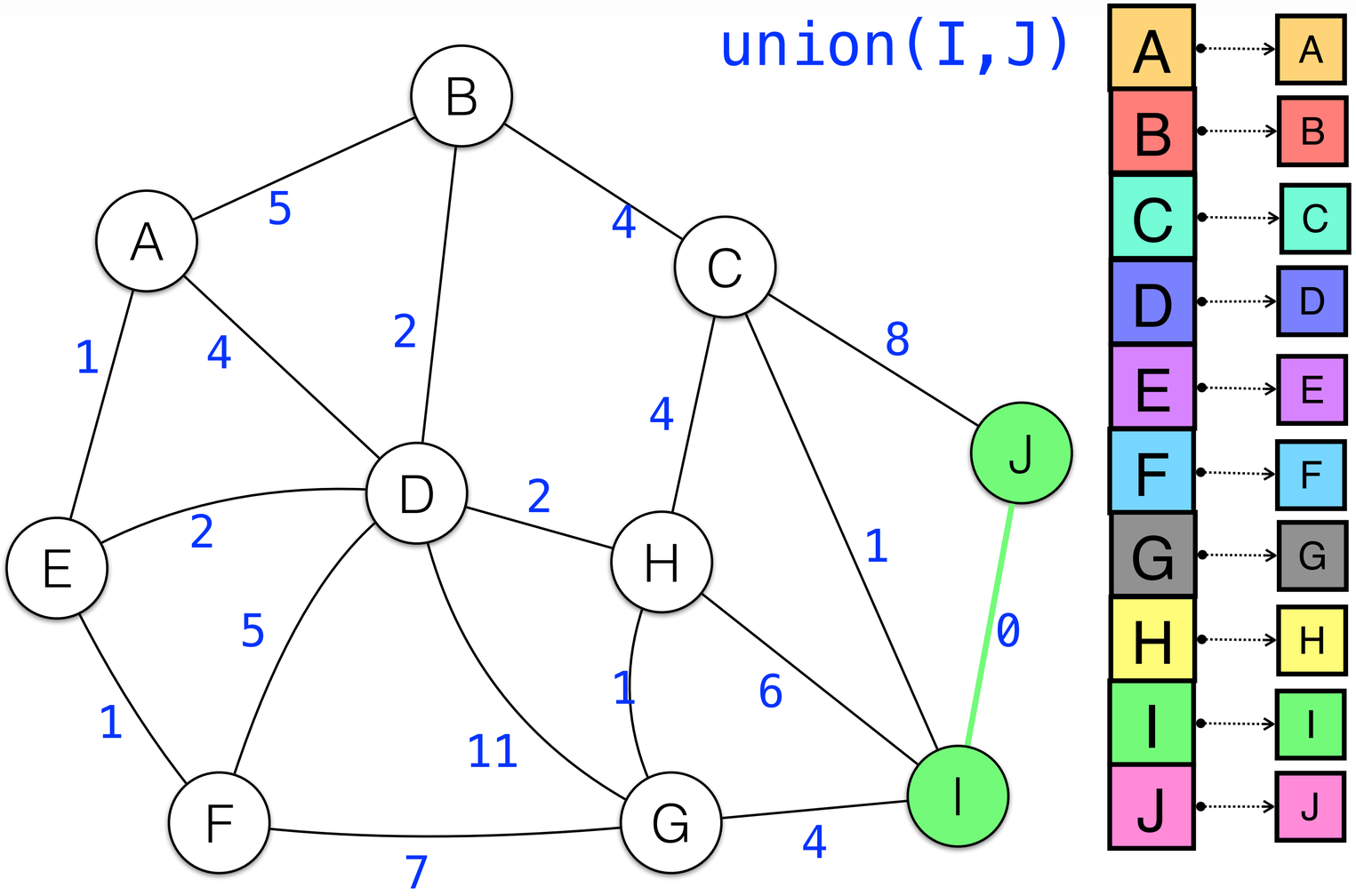
	I-J	A-E	C-I	E-F	G-H	B-D	C-J	D-E	D-H	A-D	B-C	C-H	G-I	A-B	D-F	H-I	F-G	D-G
d	0	1	1	1	1	2	2	2	2	4	4	4	4	5	5	6	7	11





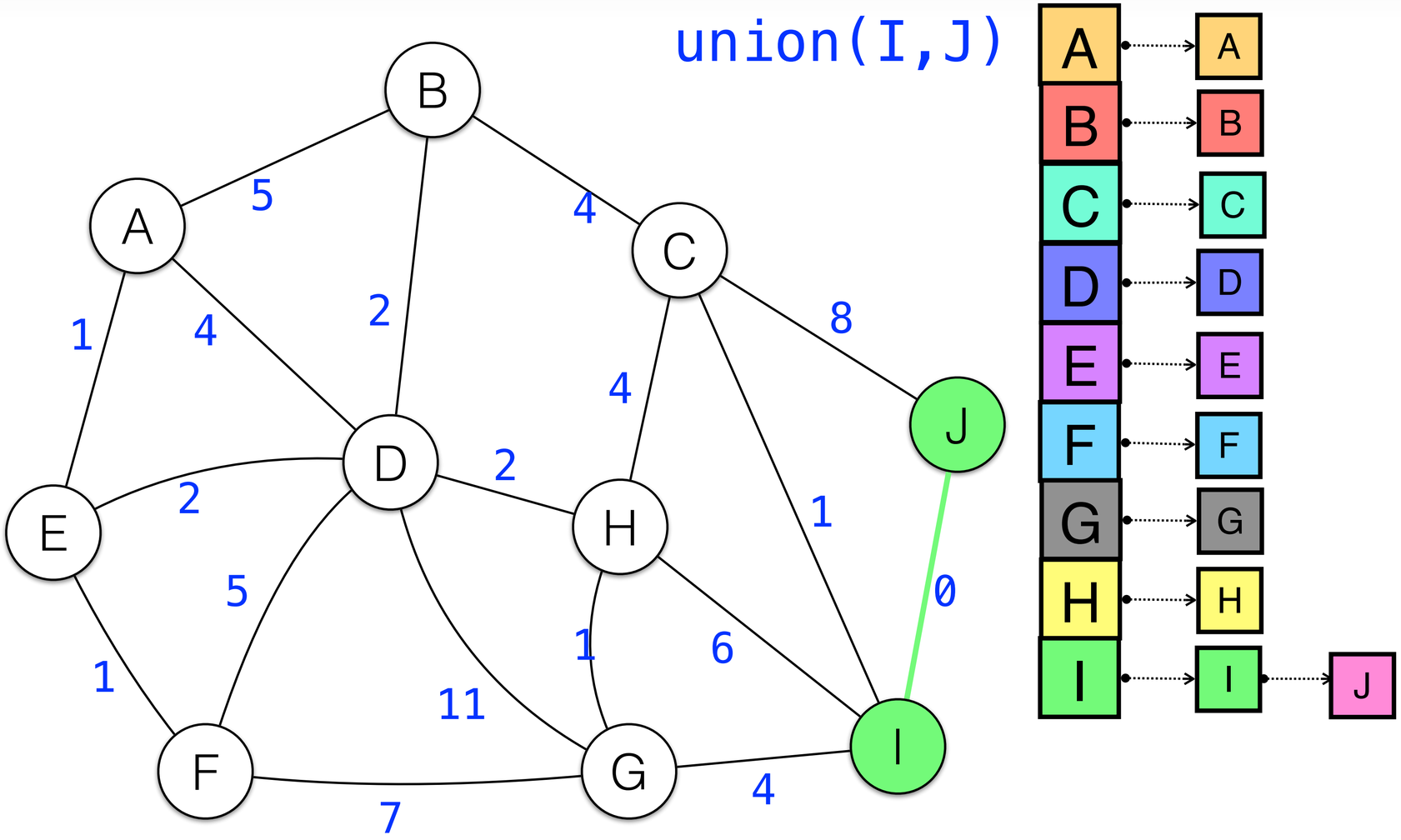
Kruskal MST

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



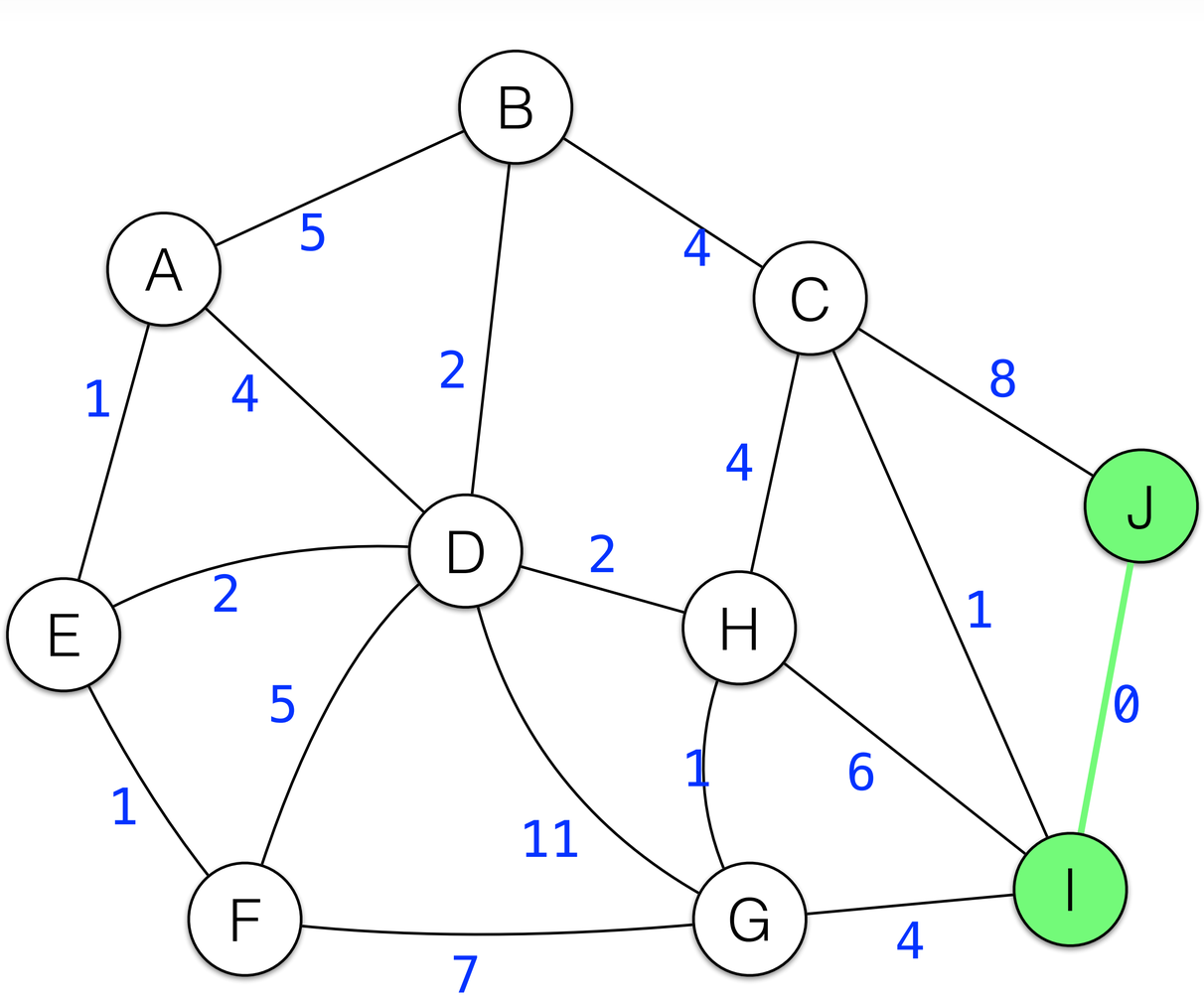
Kruskal MST

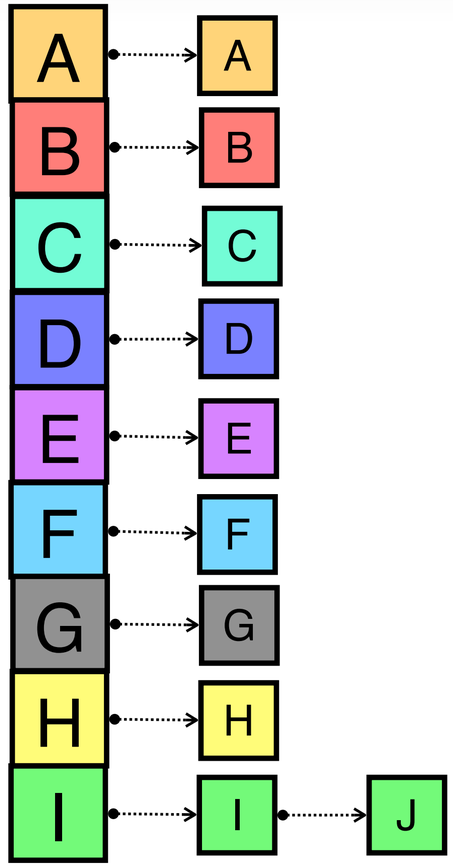
	I-J	A-E	C-I	E-F	G-H	B-D	C-J	D-E	D-H	A-D	B-C	C-H	G-I	A-B	D-F	H-I	F-G	D-G
d	0	1	1	1	1	2	2	2	2	4	4	4	4	5	5	6	7	11



Kruskal MST

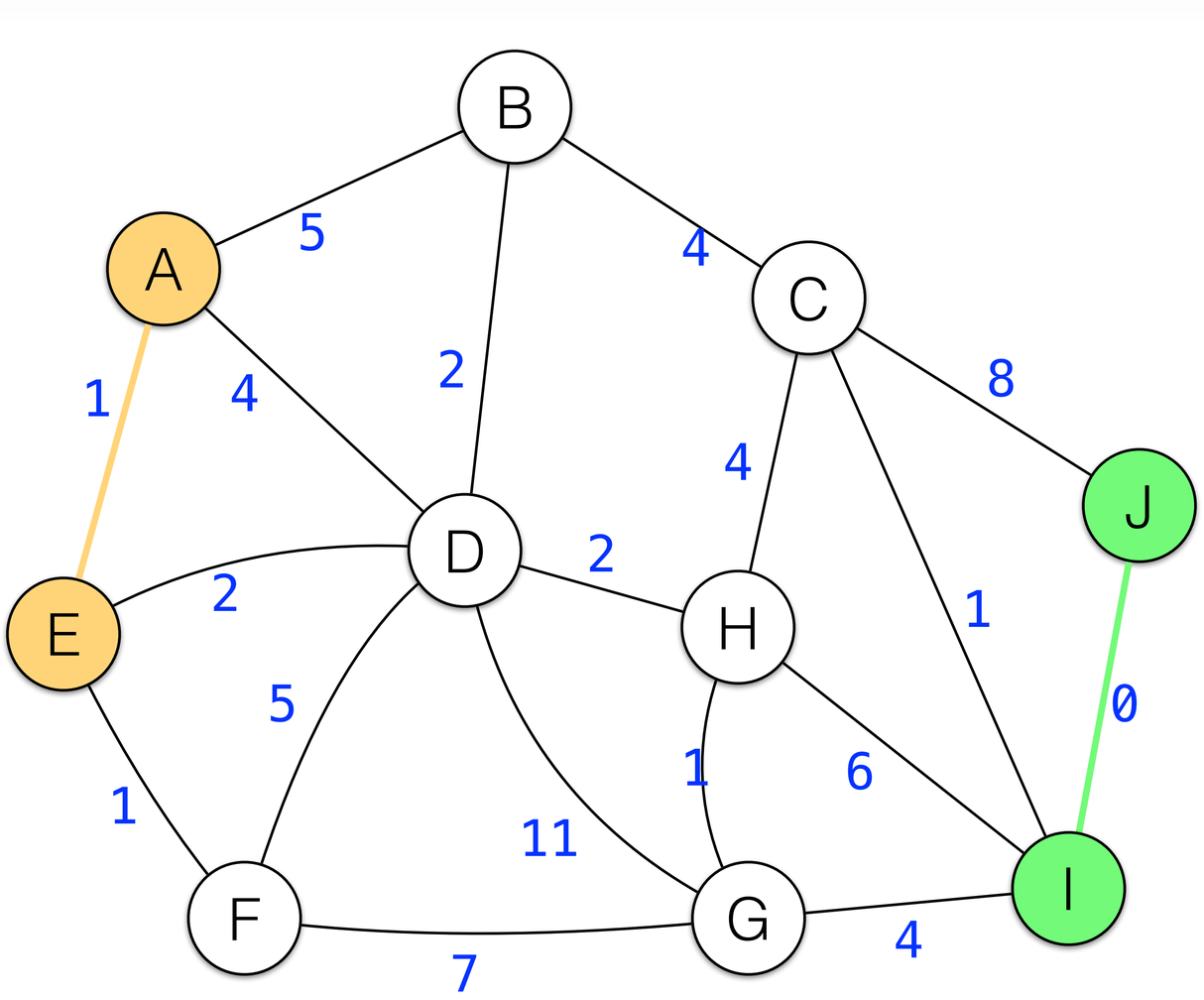
	I-J	A-E	C-I	E-F	G-H	B-D	C-J	D-E	D-H	A-D	B-C	C-H	G-I	A-B	D-F	H-I	F-G	D-G
d	0	1	1	1	1	2	2	2	2	4	4	4	4	5	5	6	7	11

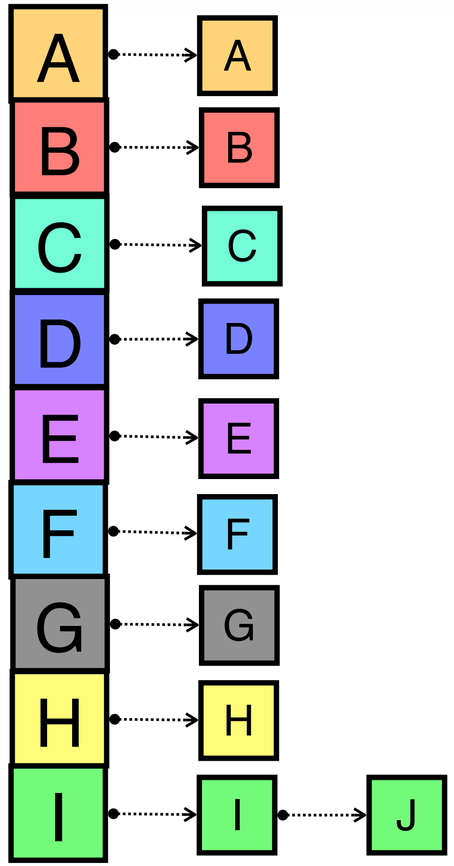




Kruskal MST

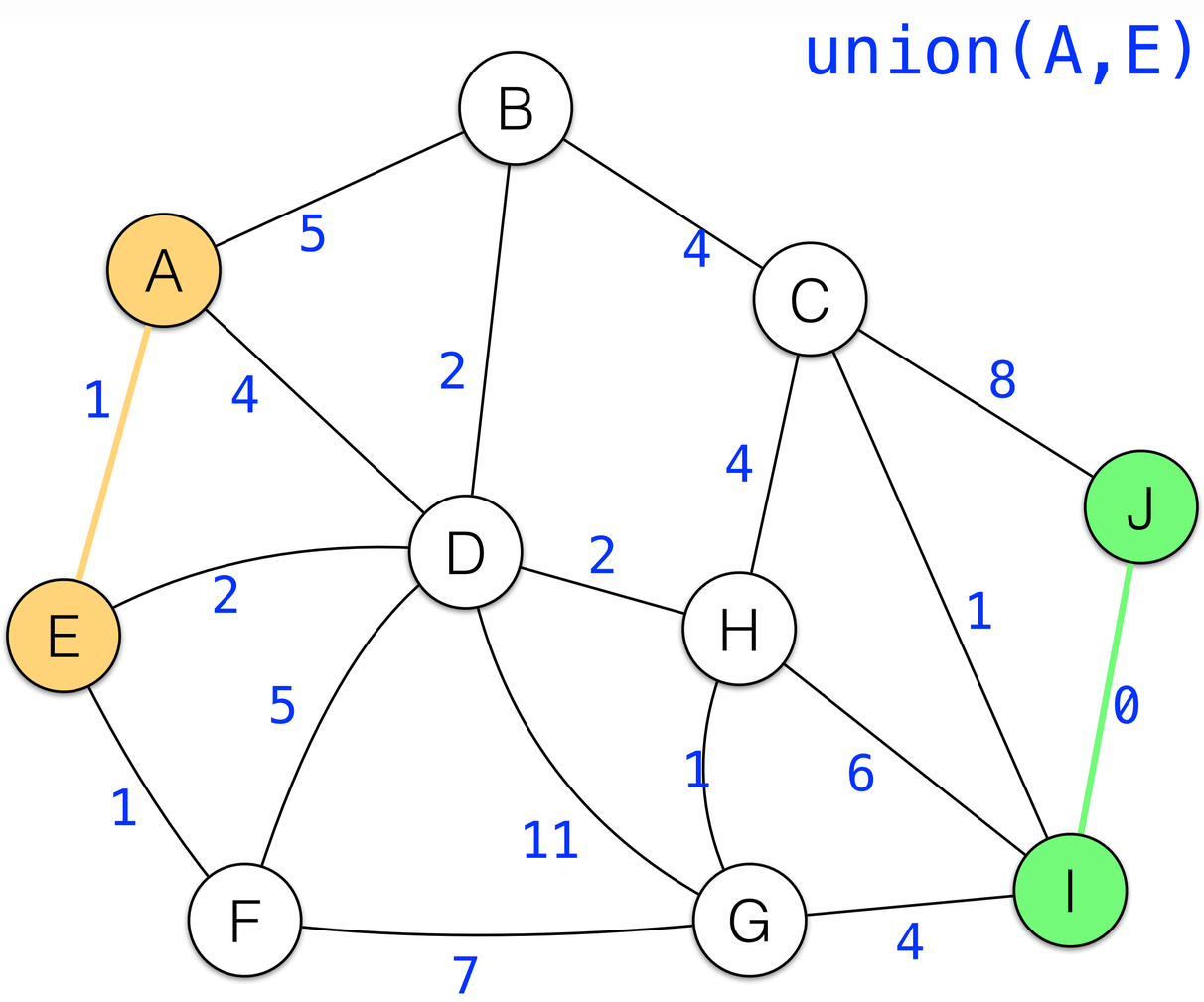
	I-J	A-E	C-I	E-F	G-H	B-D	C-J	D-E	D-H	A-D	B-C	C-H	G-I	A-B	D-F	H-I	F-G	D-G
d	0	1	1	1	1	2	2	2	2	4	4	4	4	5	5	6	7	11

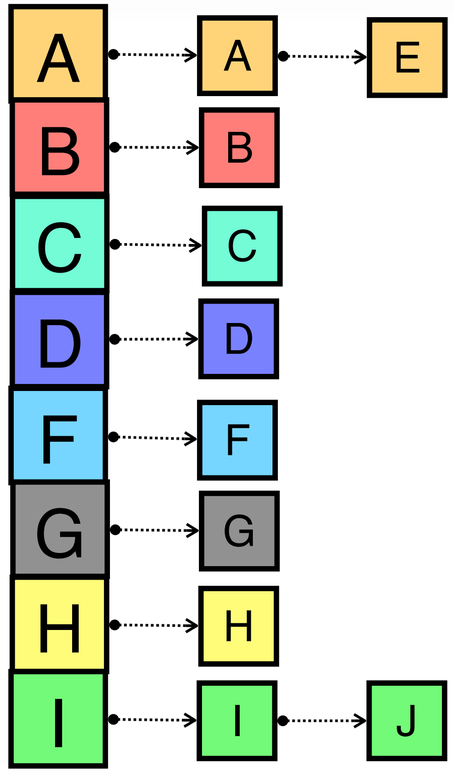




Kruskal MST

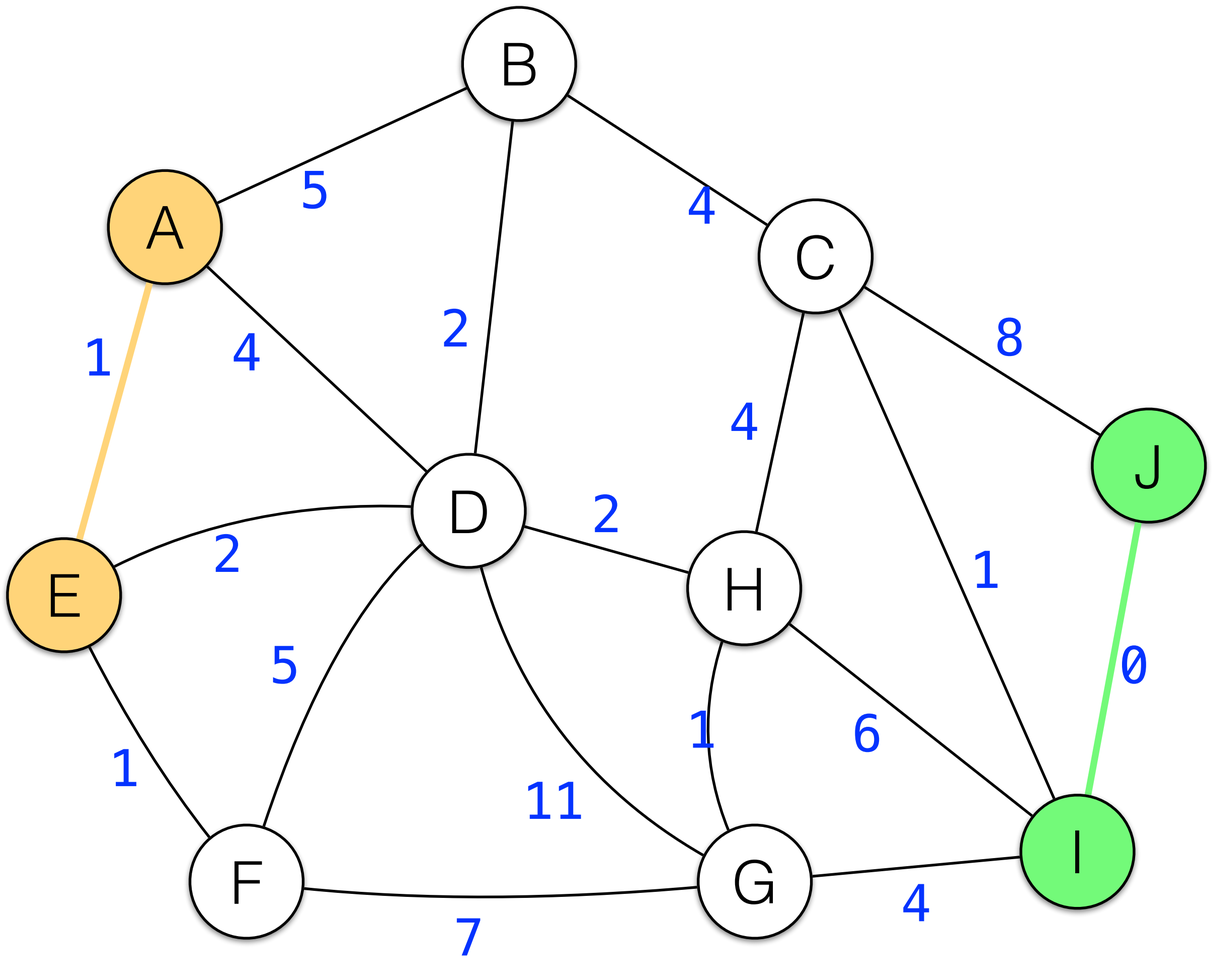
	I-J	A-E	C-I	E-F	G-H	B-D	C-J	D-E	D-H	A-D	B-C	C-H	G-I	A-B	D-F	H-I	F-G	D-G
d	0	1	1	1	1	2	2	2	2	4	4	4	4	5	5	6	7	11

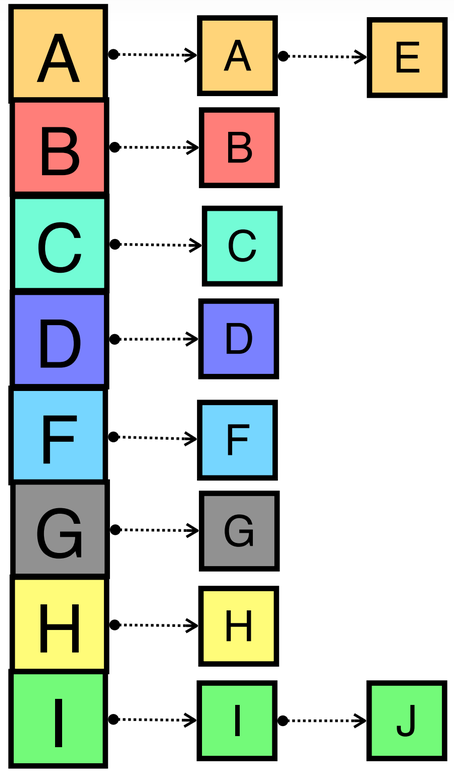




Kruskal MST

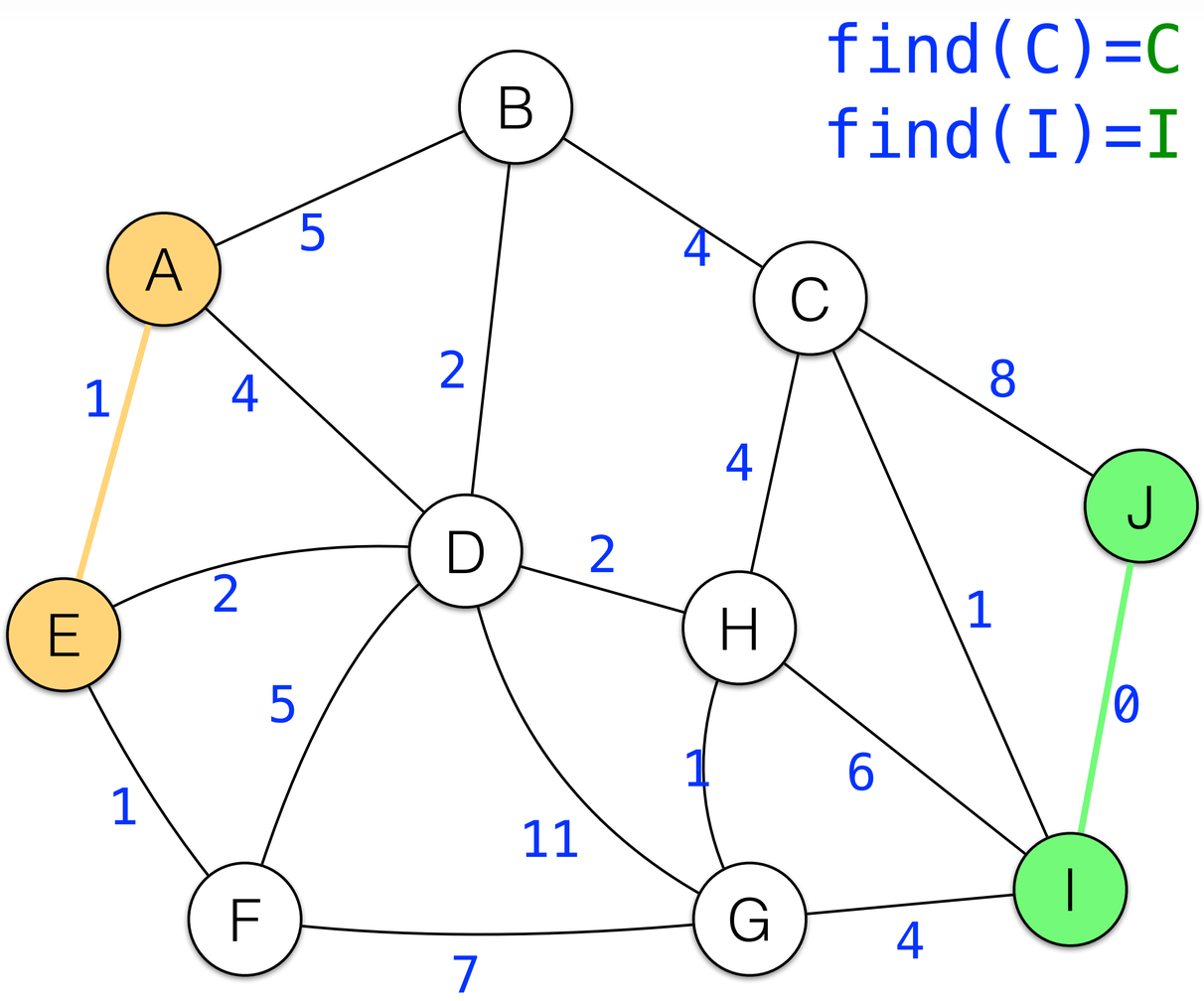
	I-J	A-E	C-I	E-F	G-H	B-D	C-J	D-E	D-H	A-D	B-C	C-H	G-I	A-B	D-F	H-I	F-G	D-G
d	0	1	1	1	1	2	2	2	2	4	4	4	4	5	5	6	7	11

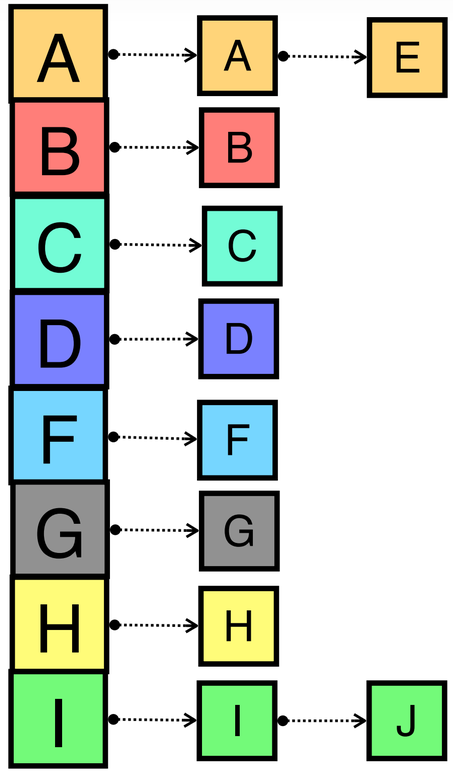




Kruskal MST

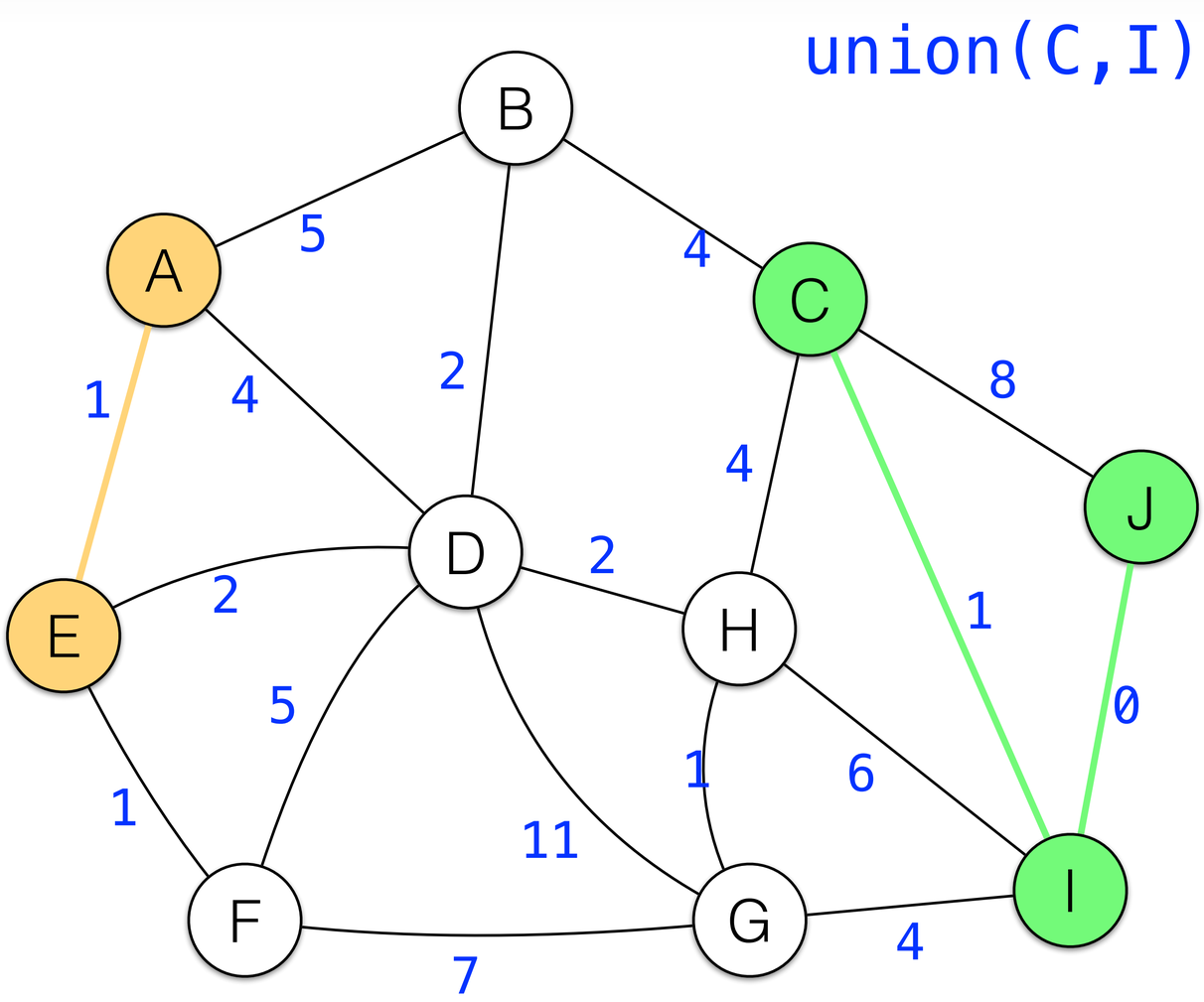
	I-J	A-E	C-I	E-F	G-H	B-D	C-J	D-E	D-H	A-D	B-C	C-H	G-I	A-B	D-F	H-I	F-G	D-G
d	0	1	1	1	1	2	2	2	2	4	4	4	4	5	5	6	7	11

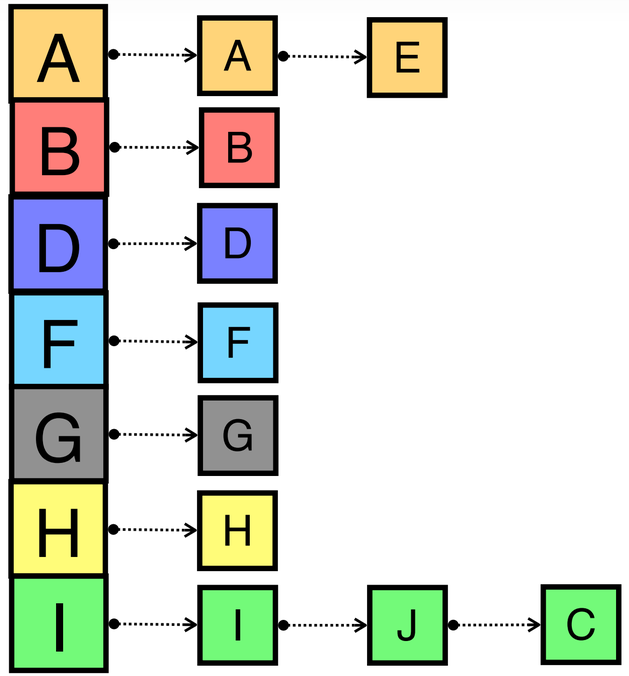




Kruskal MST

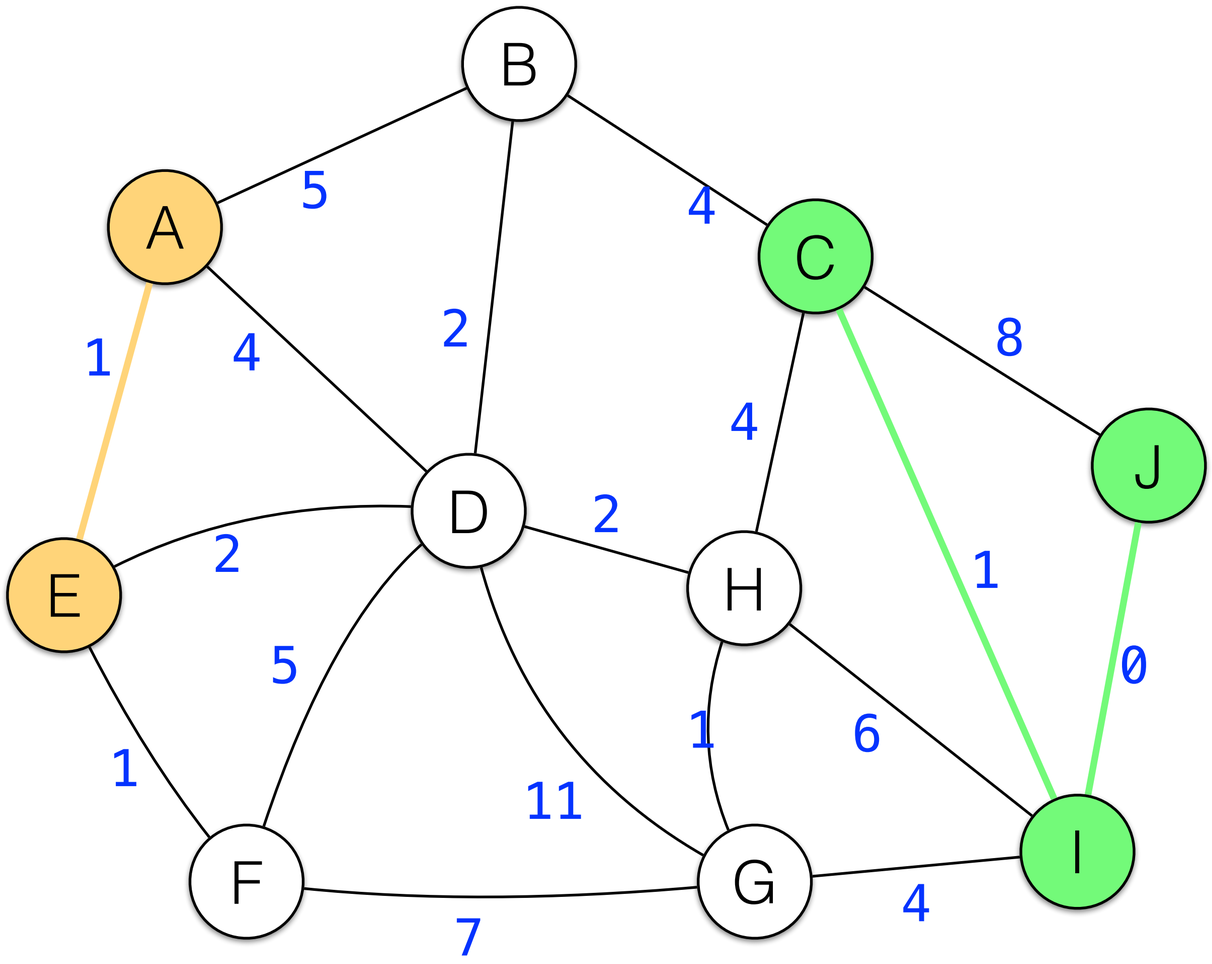
	I-J	A-E	C-I	E-F	G-H	B-D	C-J	D-E	D-H	A-D	B-C	C-H	G-I	A-B	D-F	H-I	F-G	D-G
d	0	1	1	1	1	2	2	2	2	4	4	4	4	5	5	6	7	11

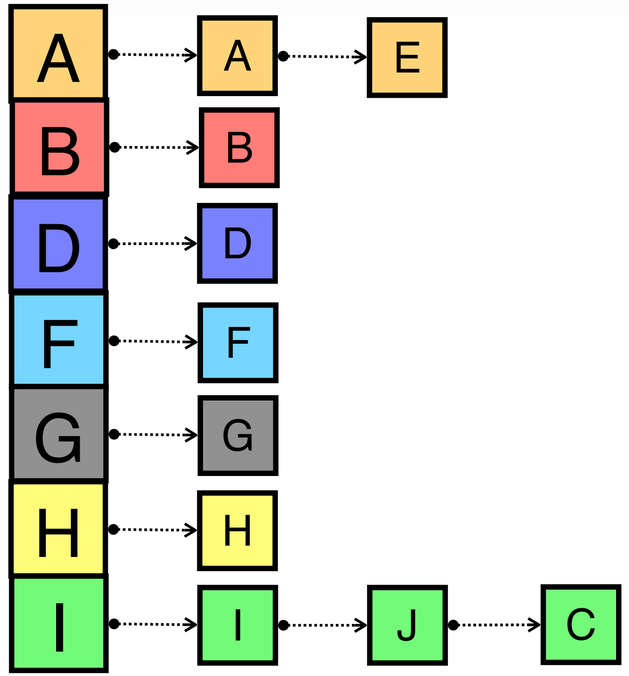




Kruskal MST

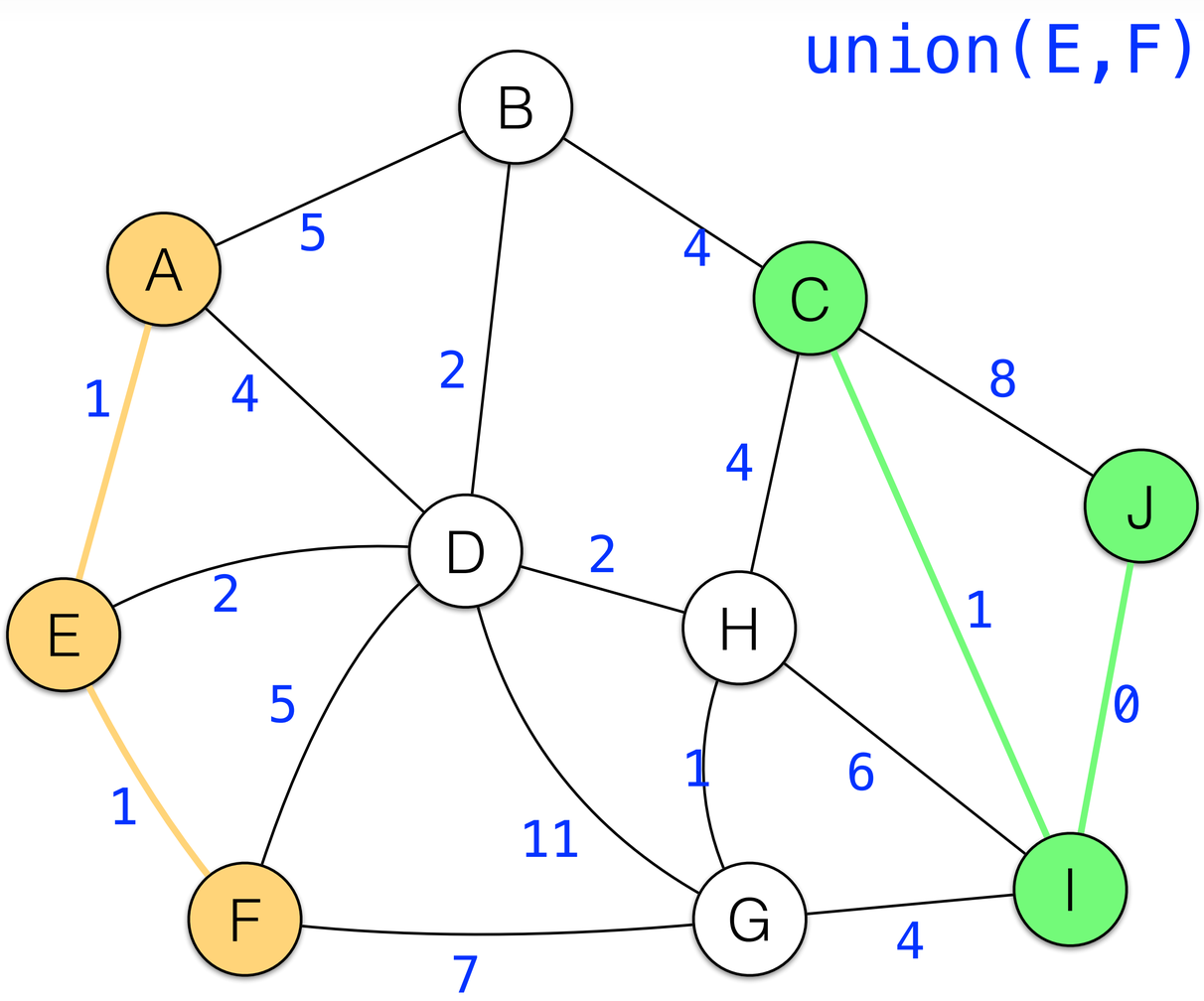
	I-J	A-E	C-I	E-F	G-H	B-D	C-J	D-E	D-H	A-D	B-C	C-H	G-I	A-B	D-F	H-I	F-G	D-G
d	0	1	1	1	1	2	2	2	2	4	4	4	4	5	5	6	7	11

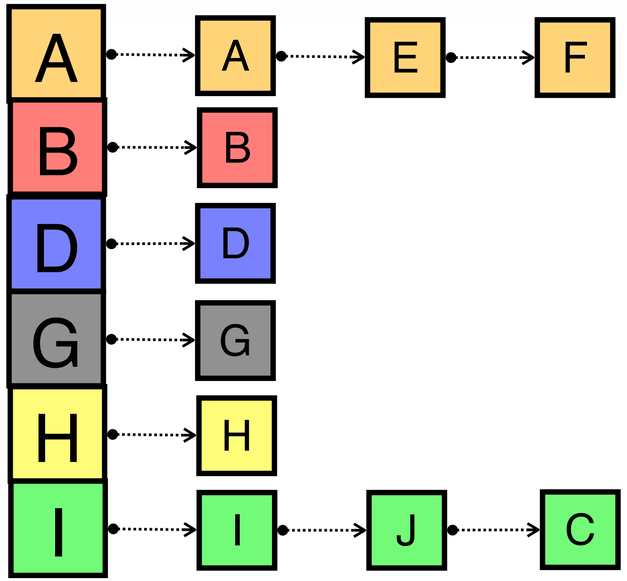




Kruskal MST

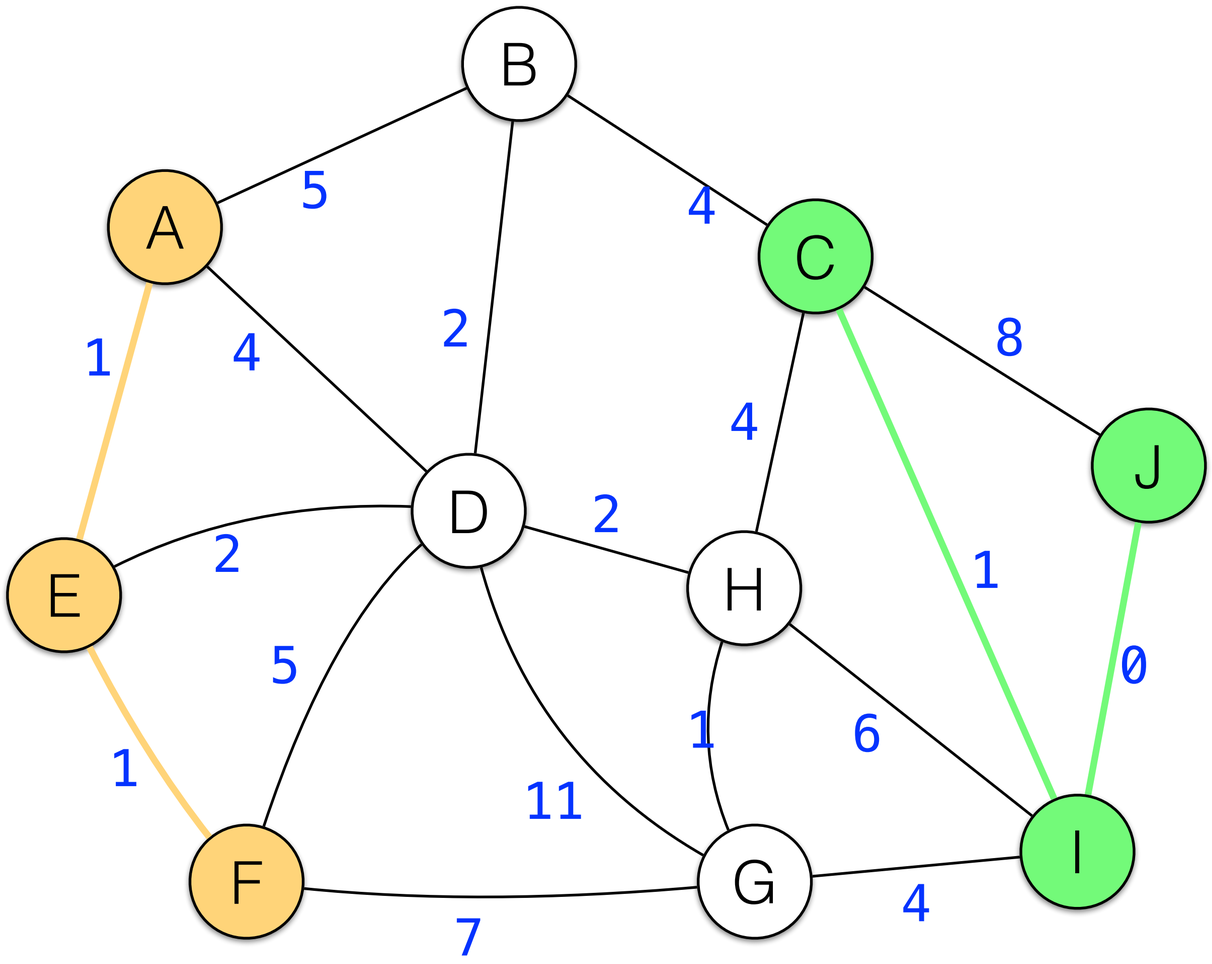
	I-J	A-E	C-I	E-F	G-H	B-D	C-J	D-E	D-H	A-D	B-C	C-H	G-I	A-B	D-F	H-I	F-G	D-G
d	0	1	1	1	1	2	2	2	2	4	4	4	4	5	5	6	7	11

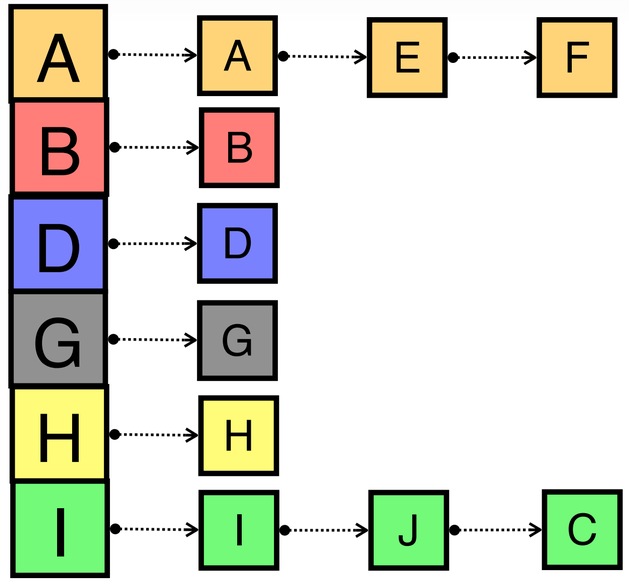




Kruskal MST

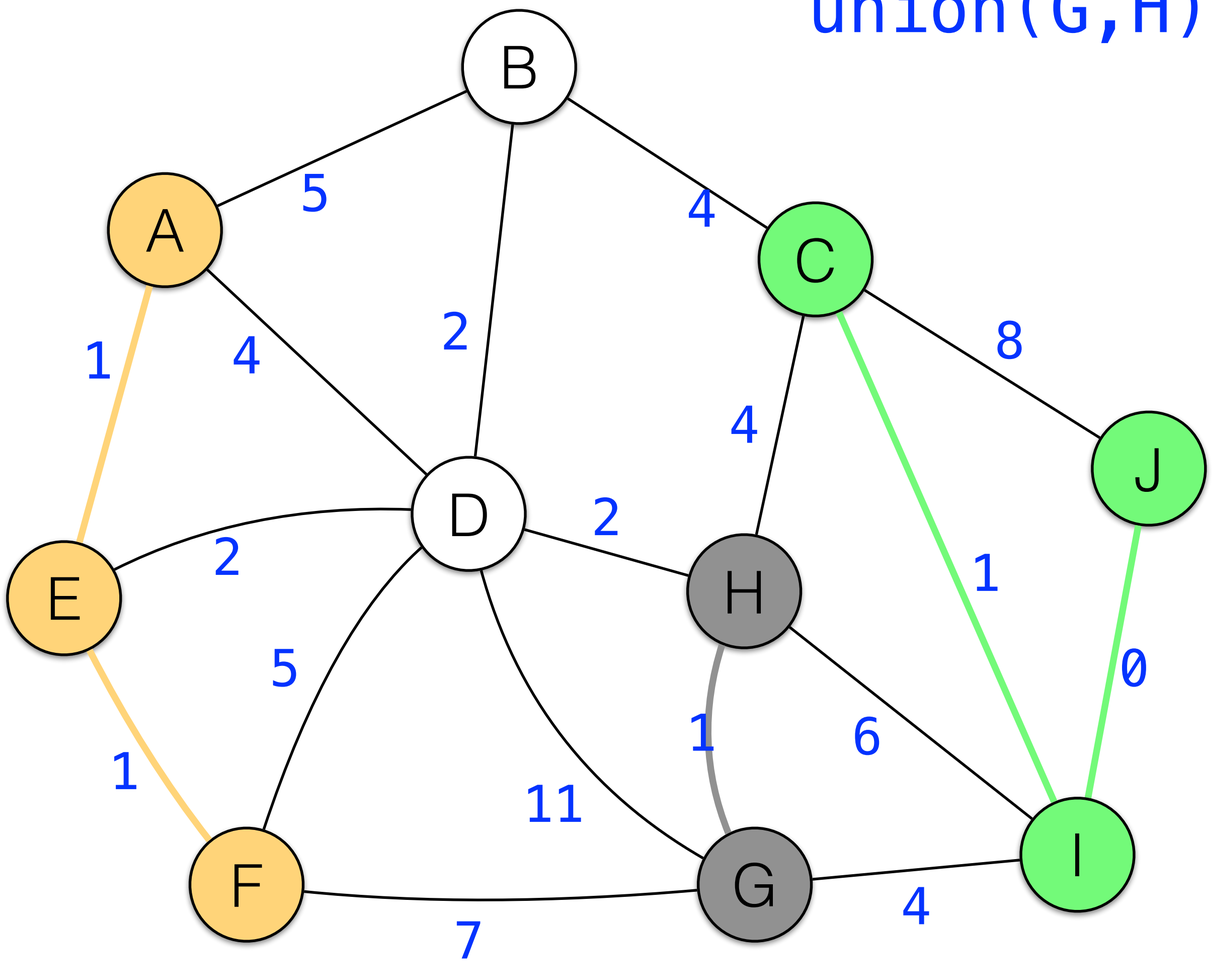
	I-J	A-E	C-I	E-F	G-H	B-D	C-J	D-E	D-H	A-D	B-C	C-H	G-I	A-B	D-F	H-I	F-G	D-G
d	0	1	1	1	1	2	2	2	2	4	4	4	4	5	5	6	7	11

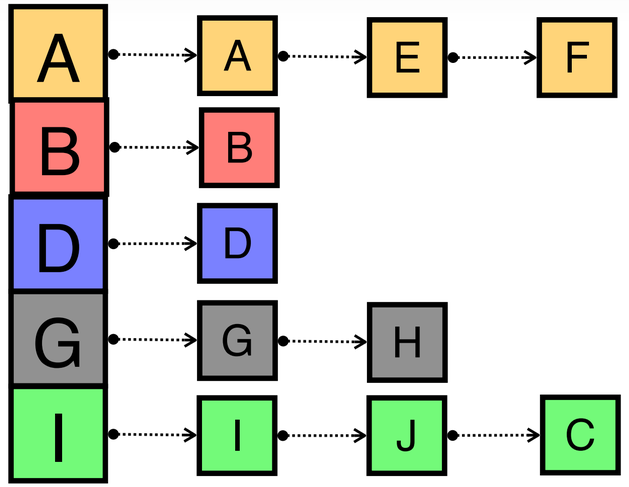




Kruskal MST

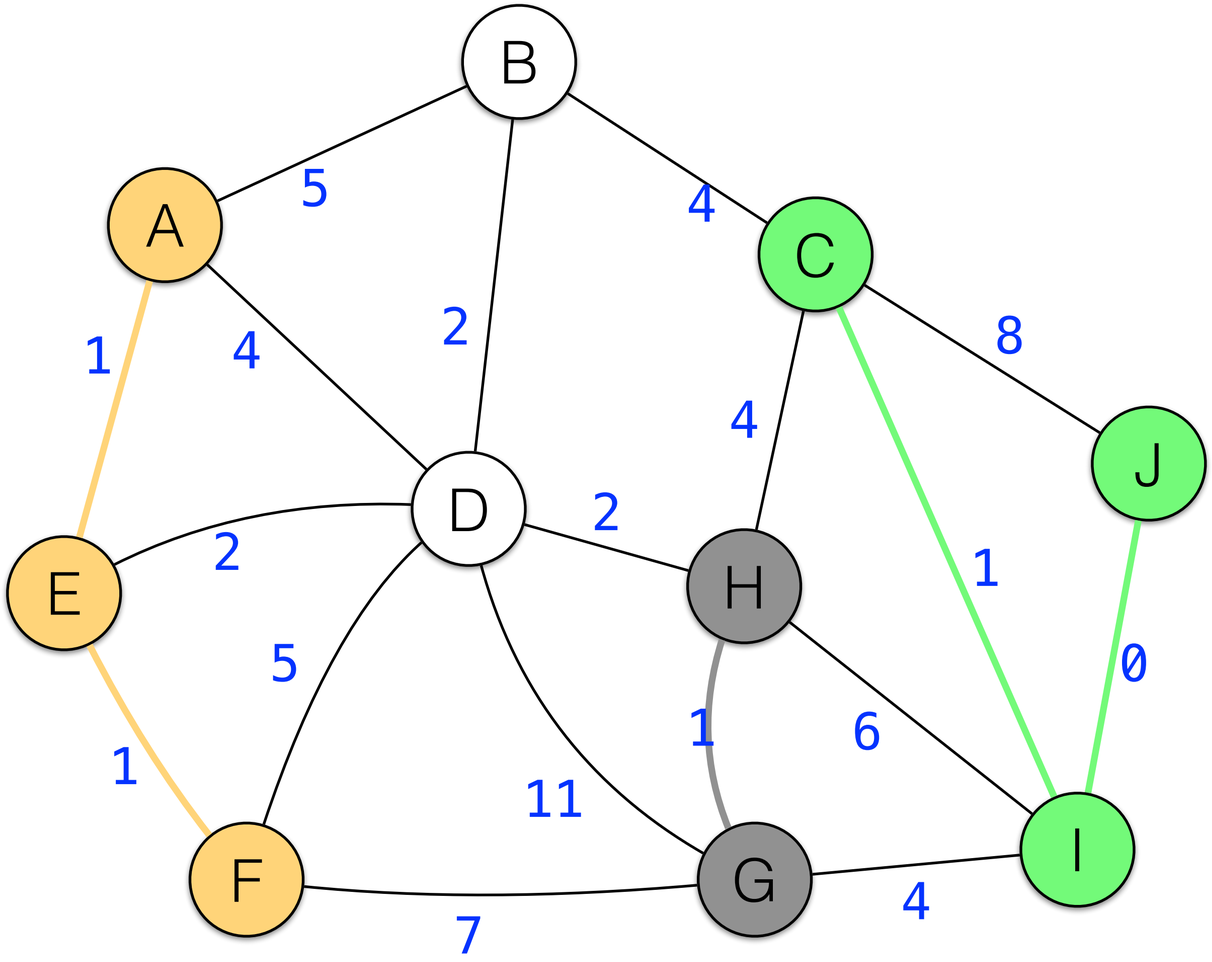
	I-J	A-E	C-I	E-F	G-H	B-D	C-J	D-E	D-H	A-D	B-C	C-H	G-I	A-B	D-F	H-I	F-G	D-G
d	0	1	1	1	1	2	2	2	2	4	4	4	4	5	5	6	7	11

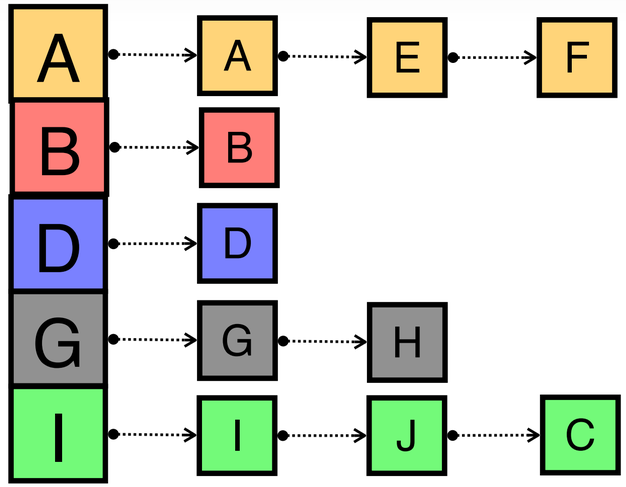




Kruskal MST

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

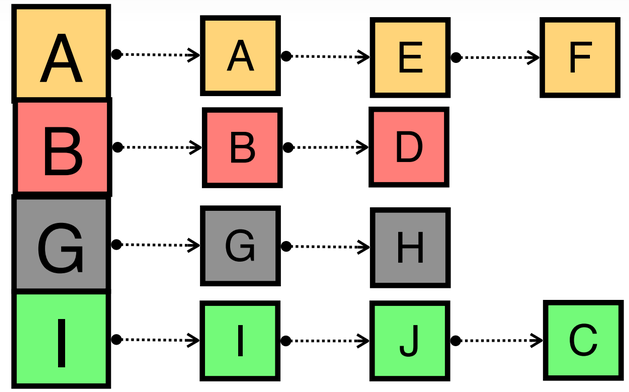


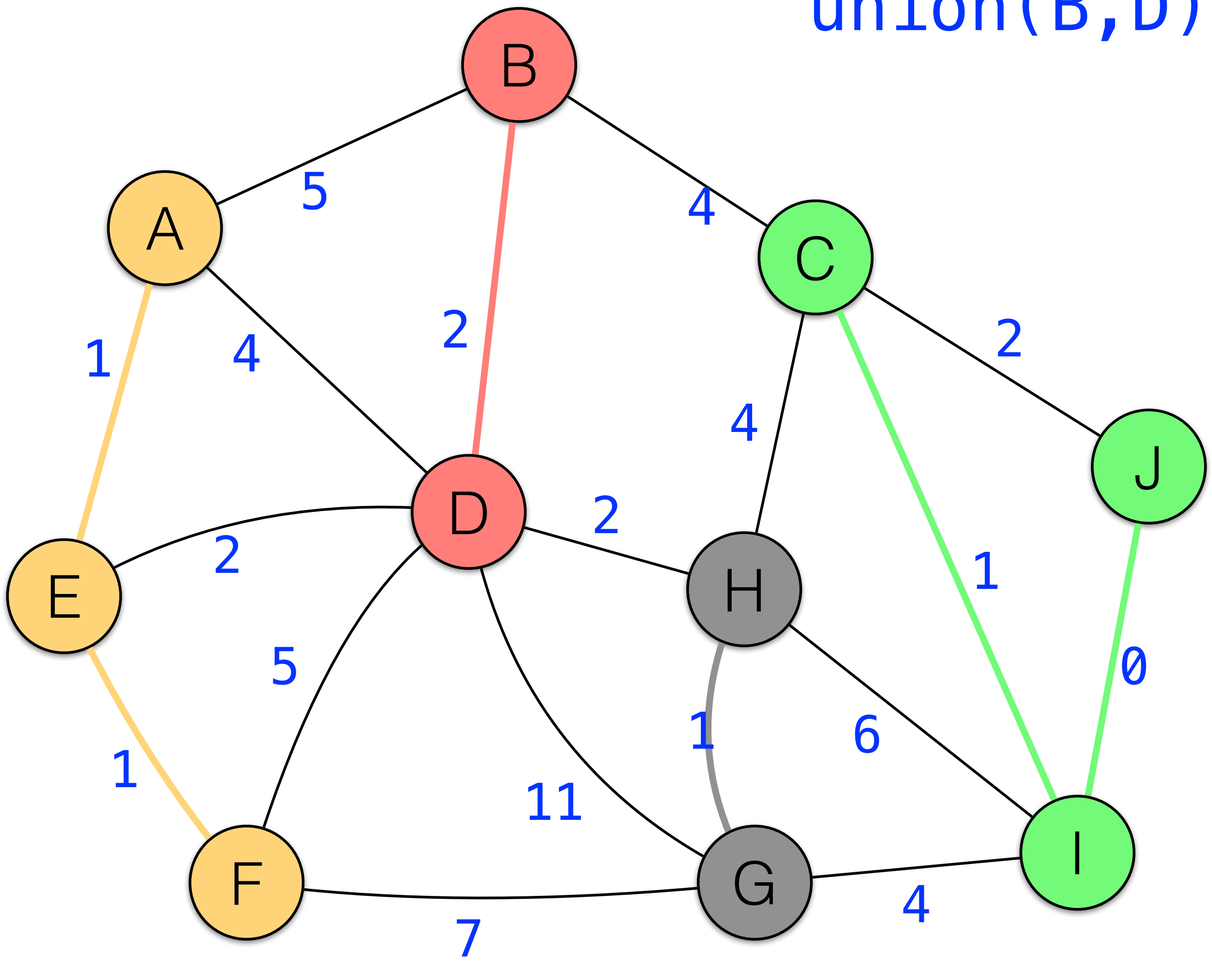


Kruskal MST

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

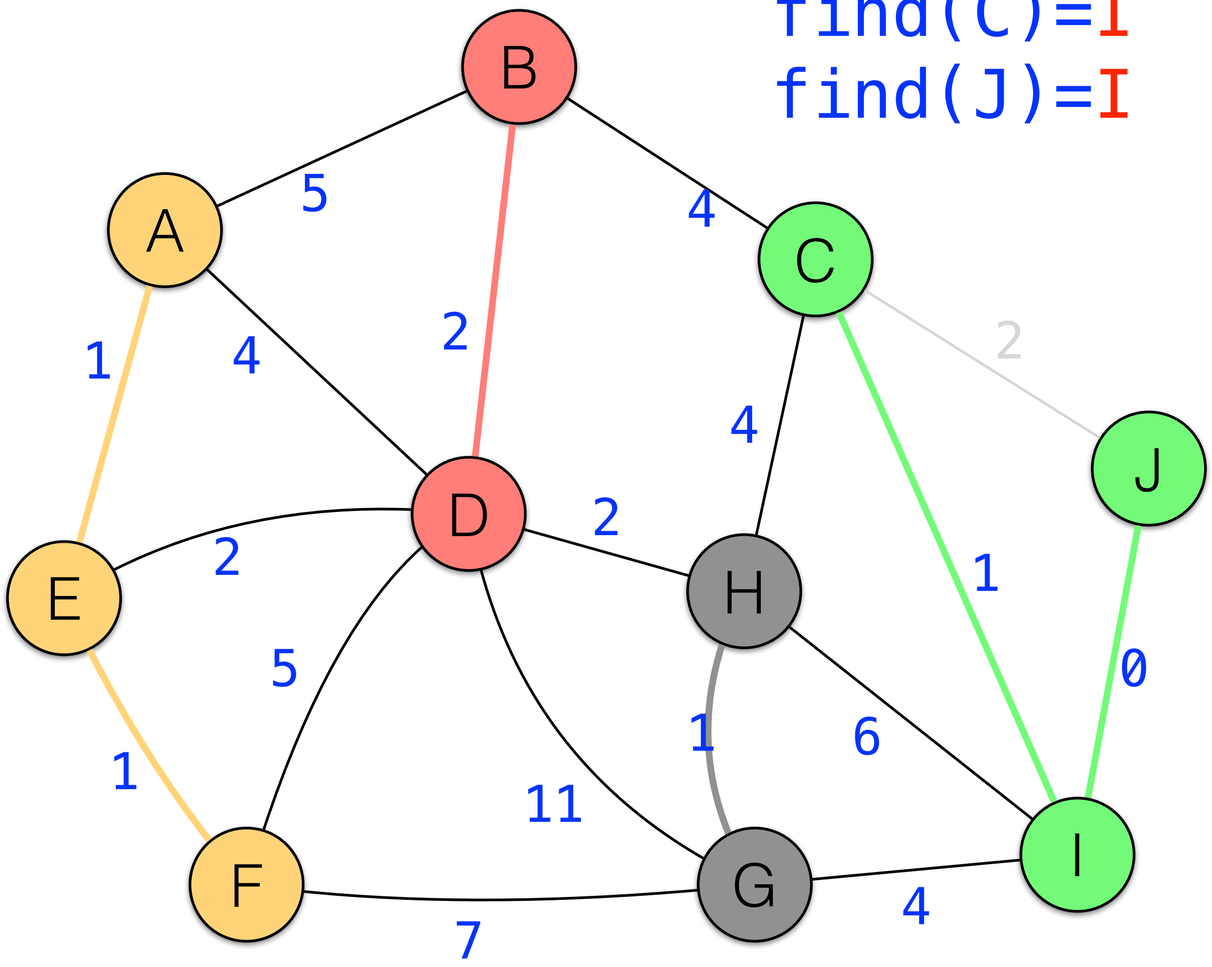
union(B,D)

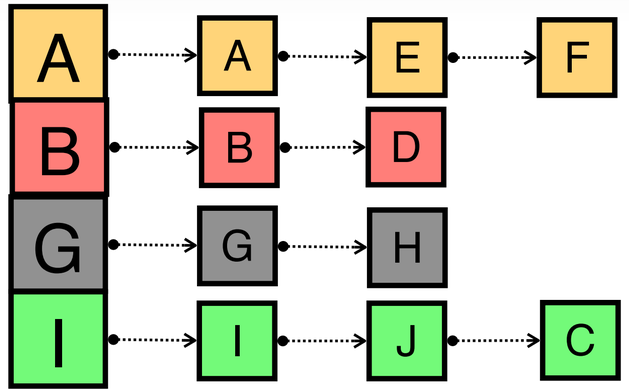




Kruskal MST

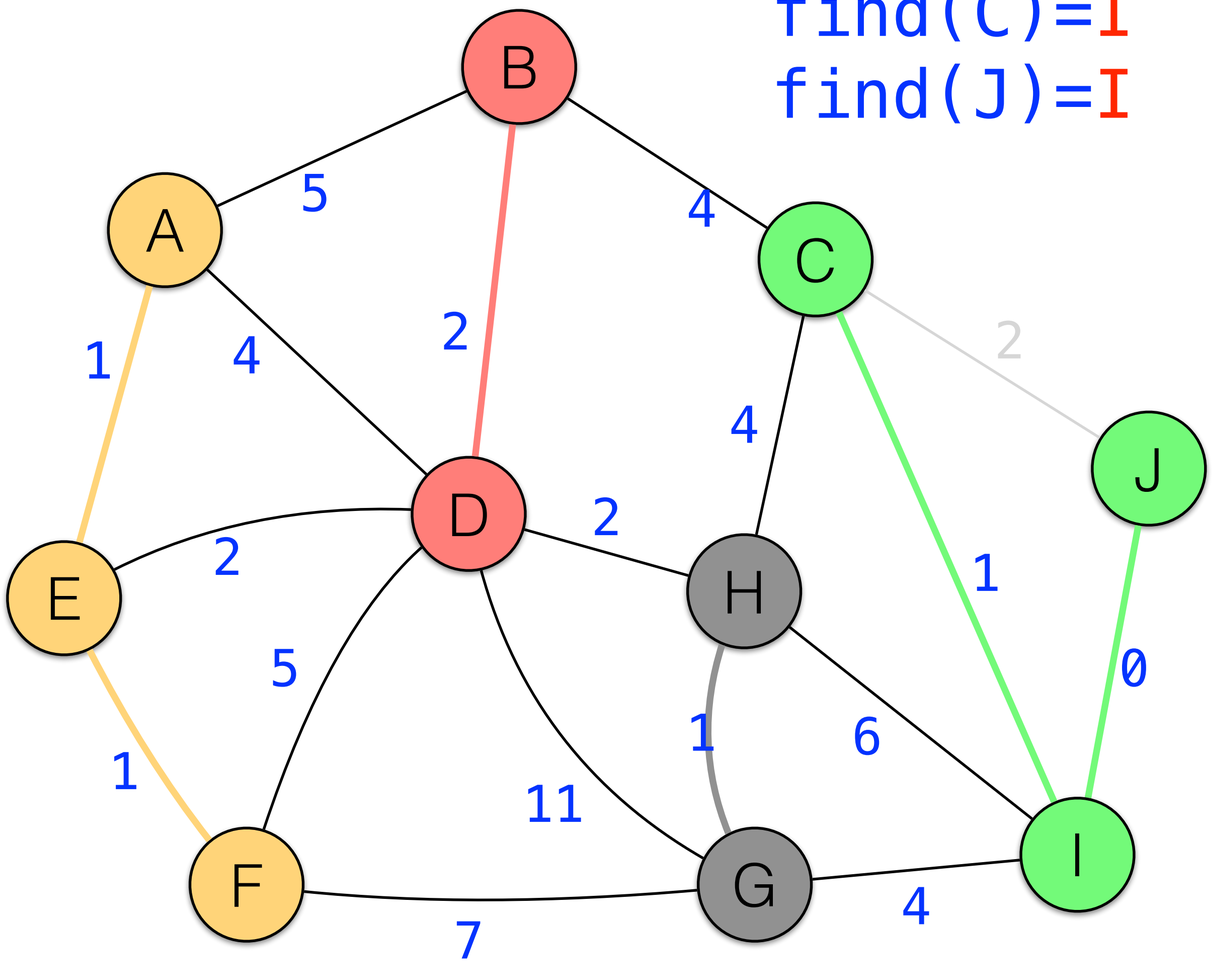
	I-J	A-E	C-I	E-F	G-H	B-D	C-J	D-E	D-H	A-D	B-C	C-H	G-I	A-B	D-F	H-I	F-G	D-G
d	0	1	1	1	1	2	2	2	2	4	4	4	4	5	5	6	7	11



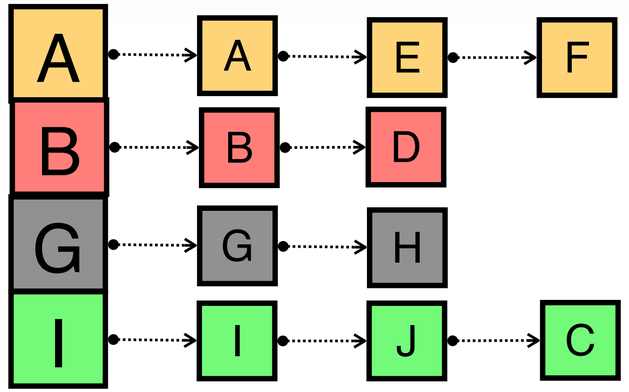


Kruskal MST

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



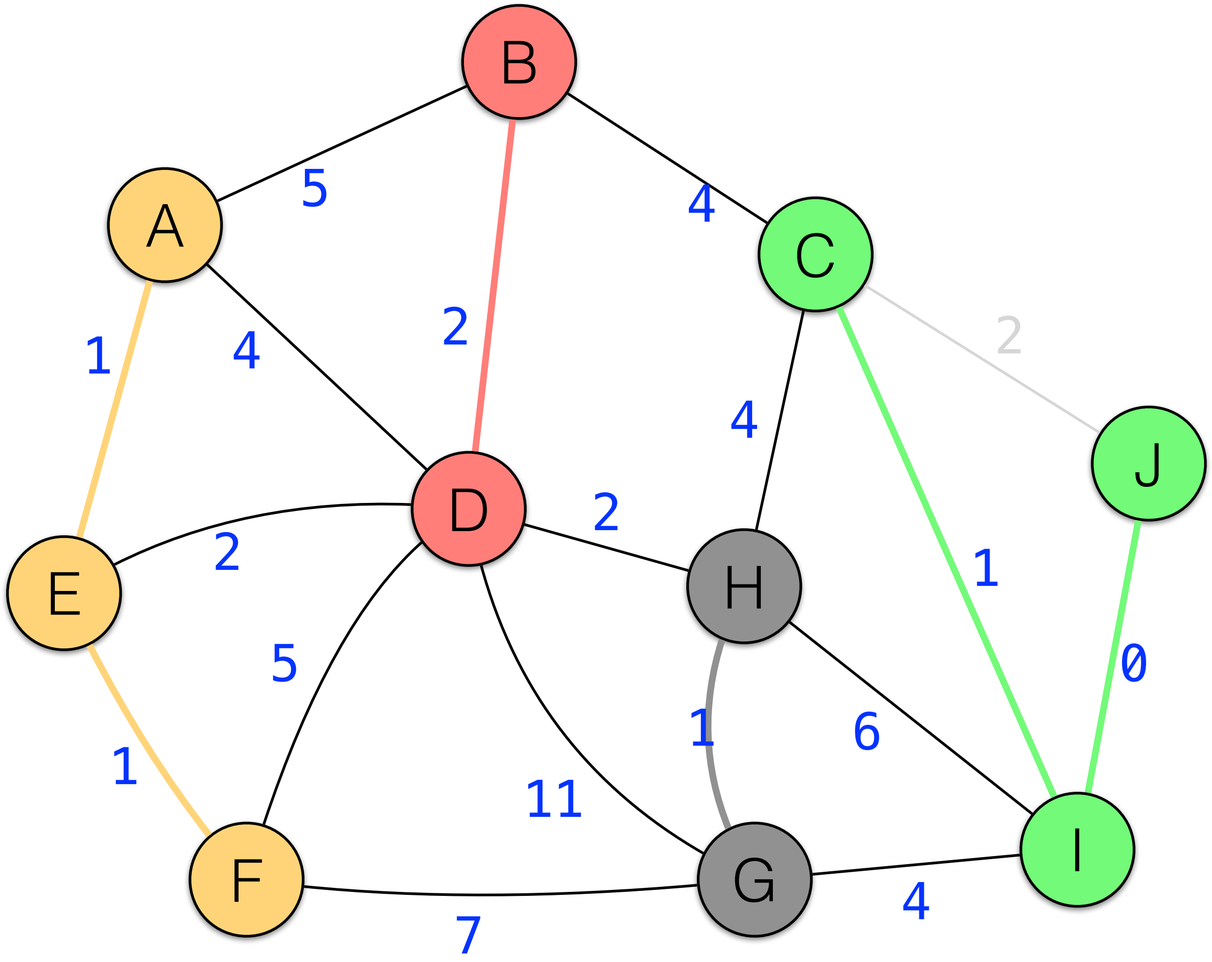
find(C)=I
find(J)=I

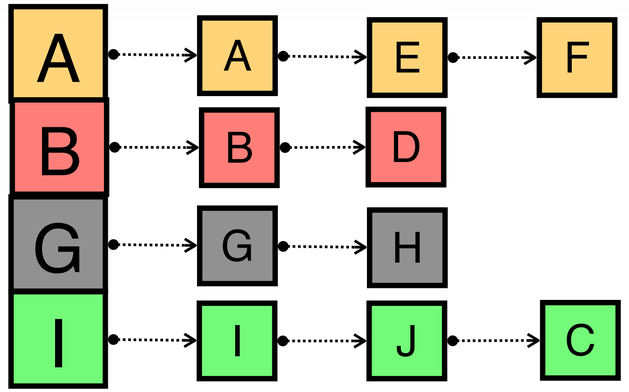


Nooooo, this generates a cycle!

Kruskal MST

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

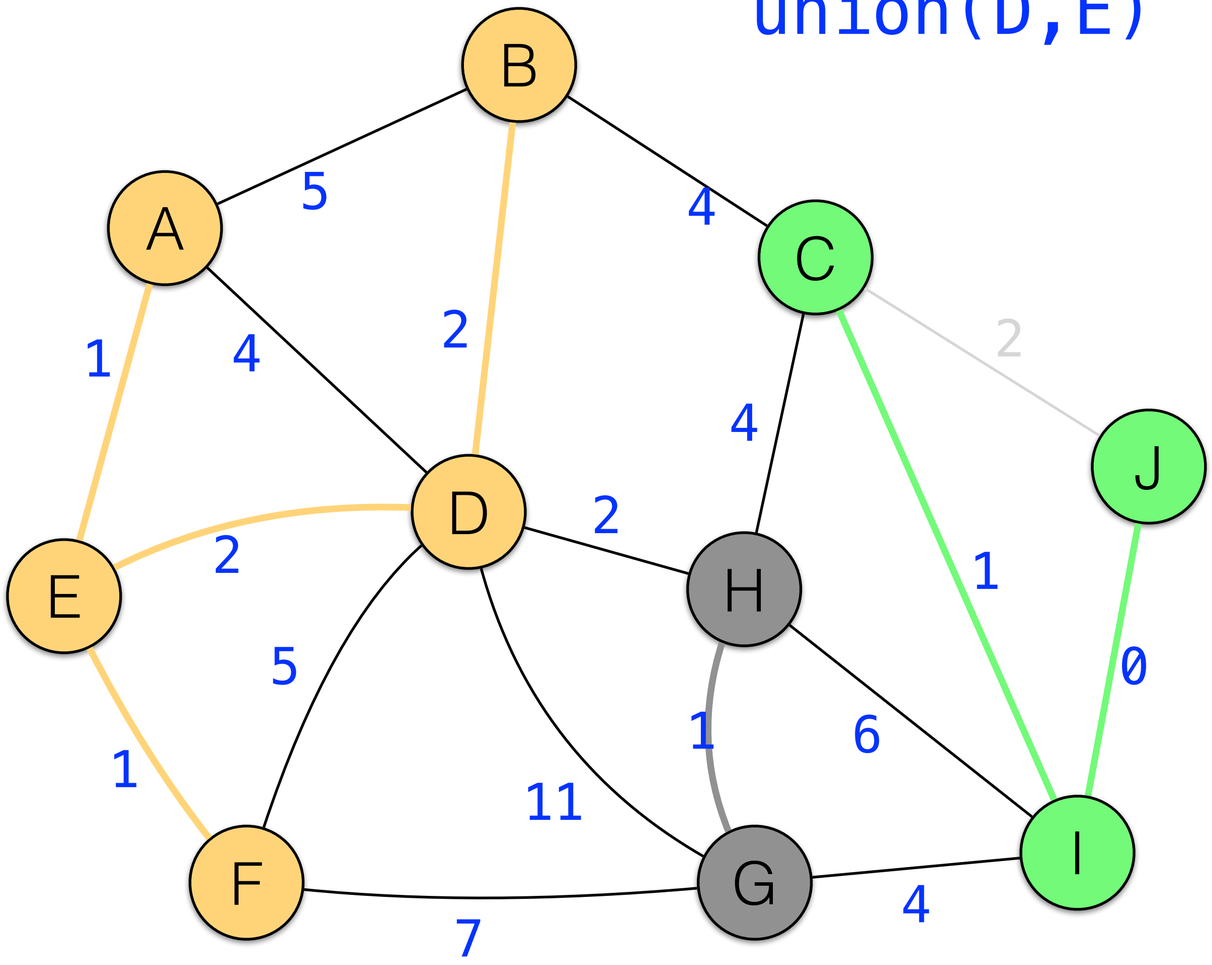


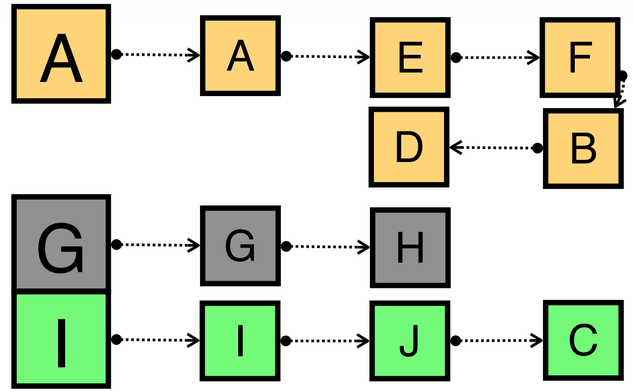


Kruskal MST

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

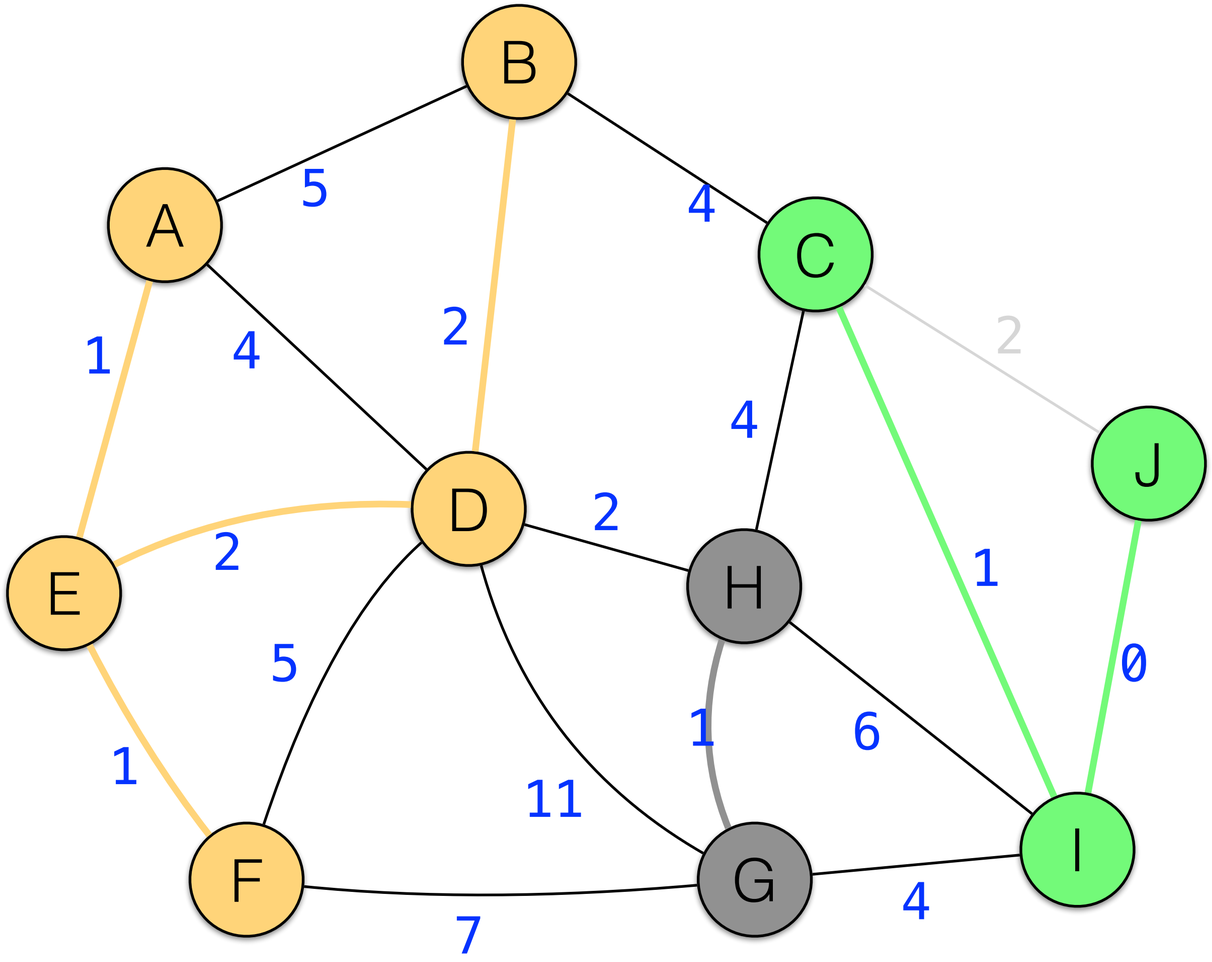
union(D,E)

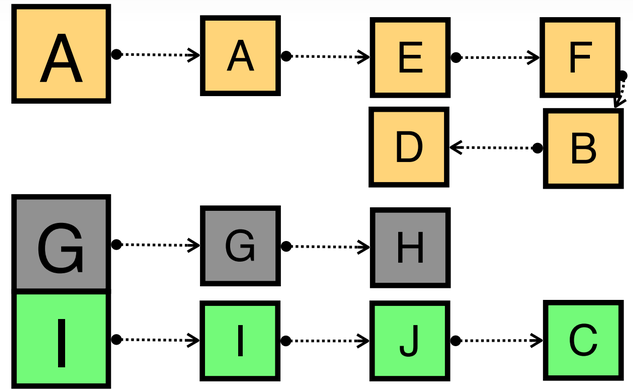




Kruskal MST

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

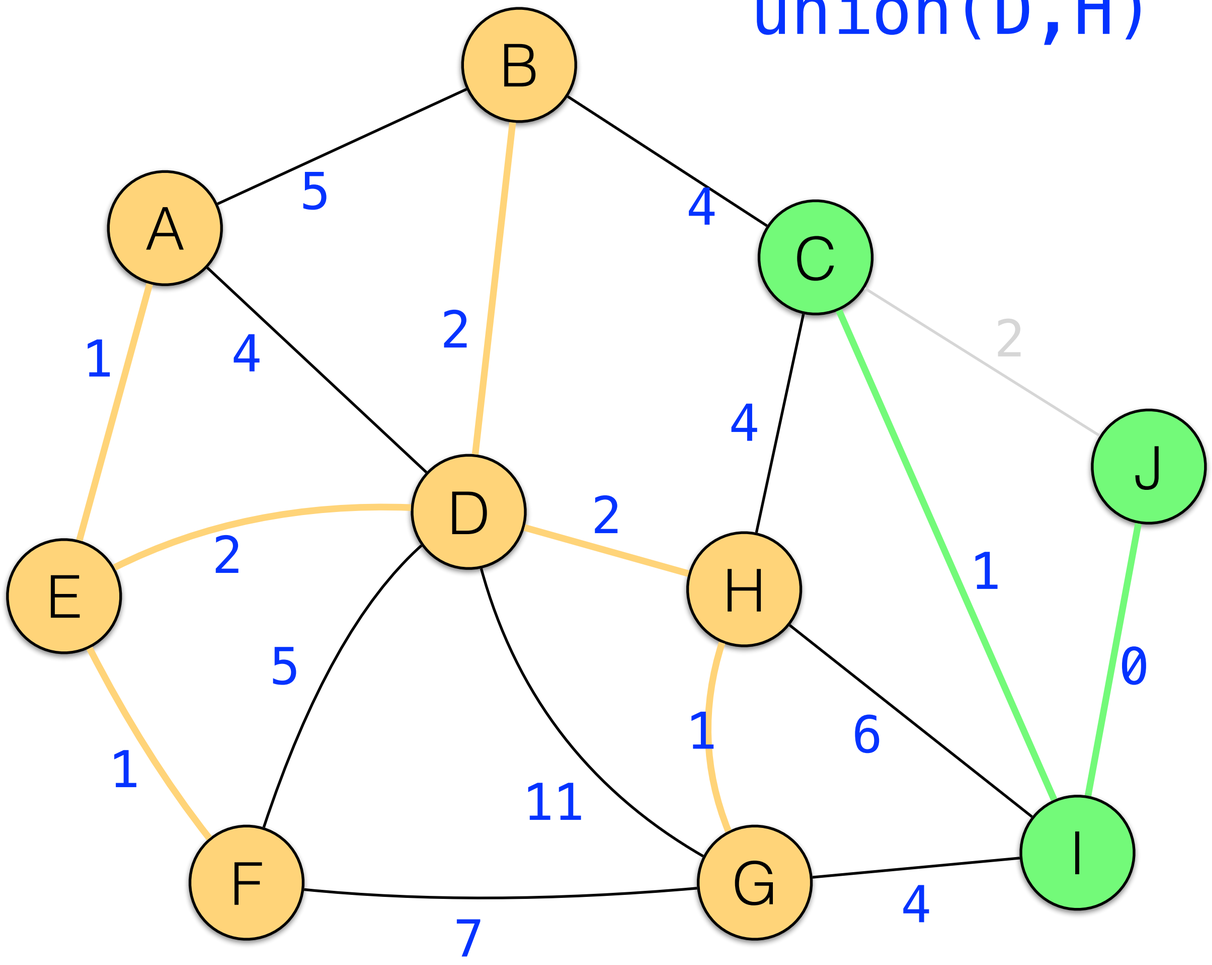


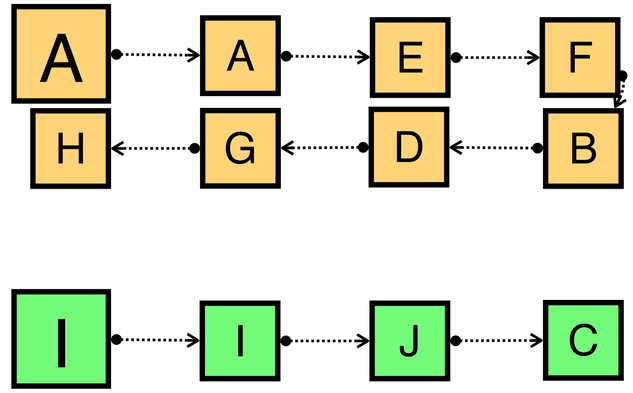


Kruskal MST

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

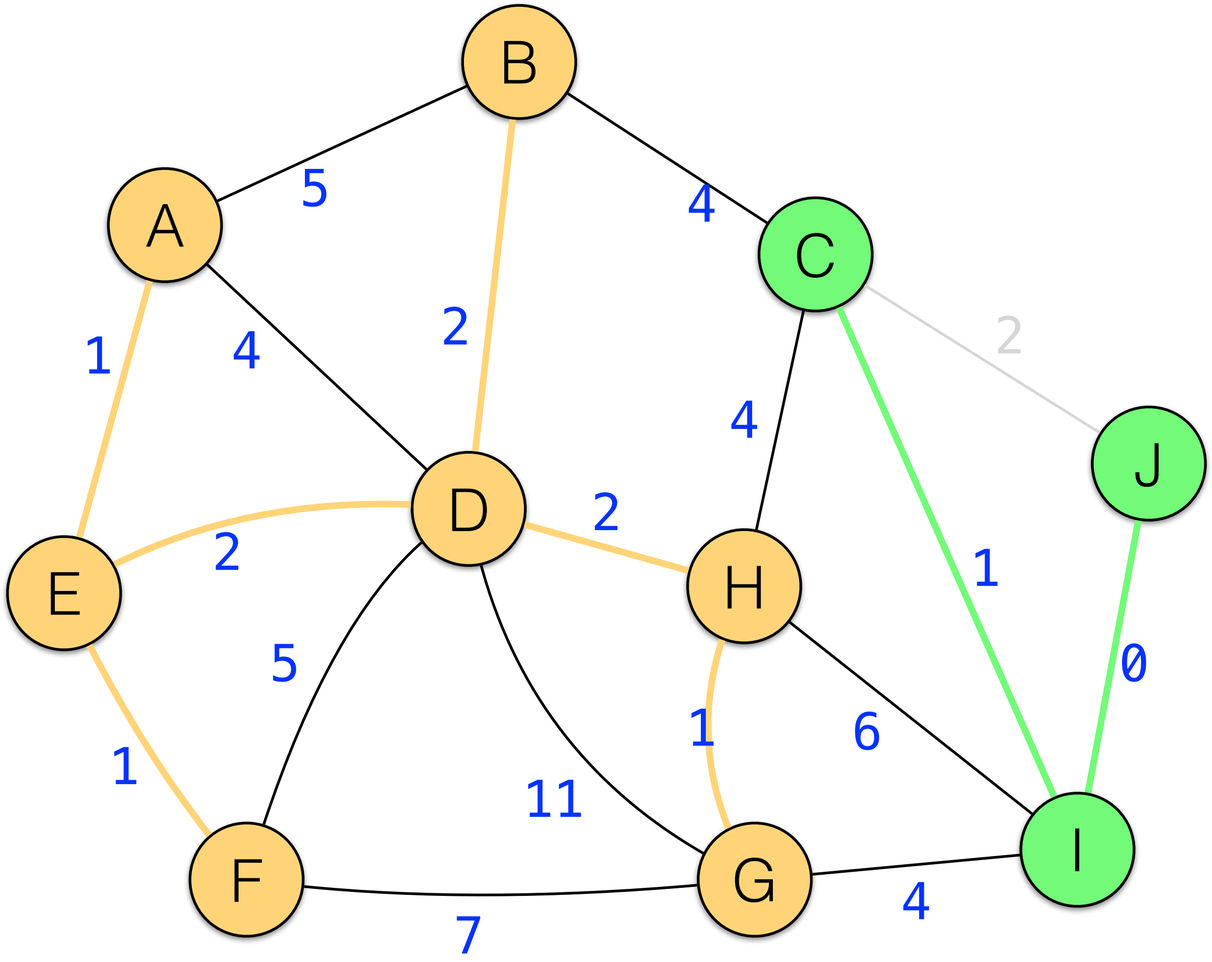
union(D,H)

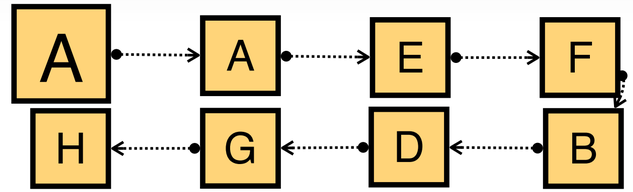


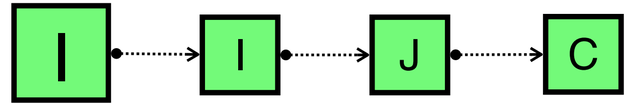


Kruskal MST

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

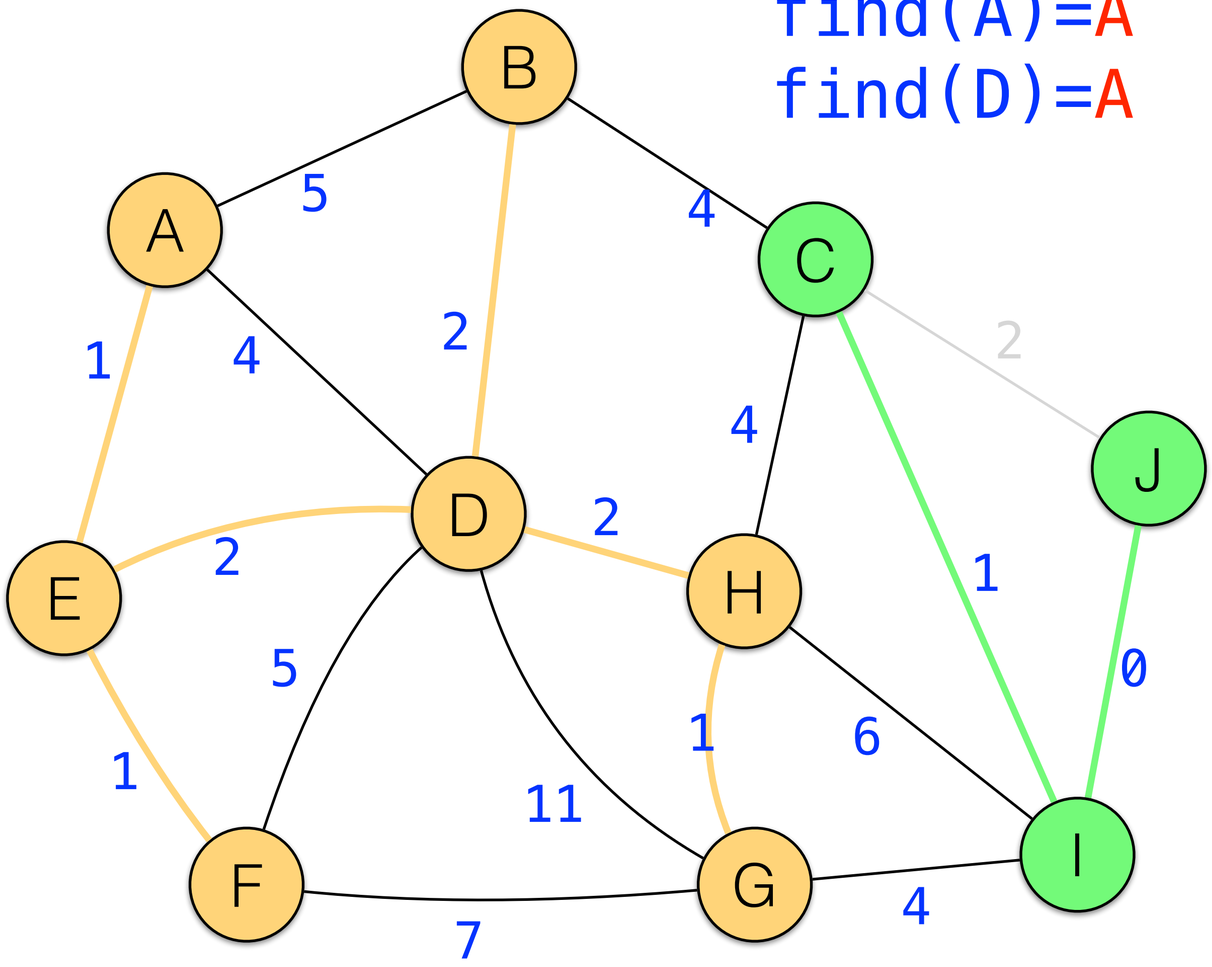




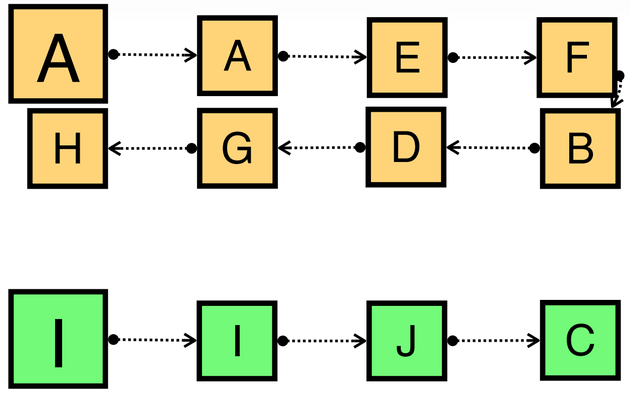


Kruskal MST

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

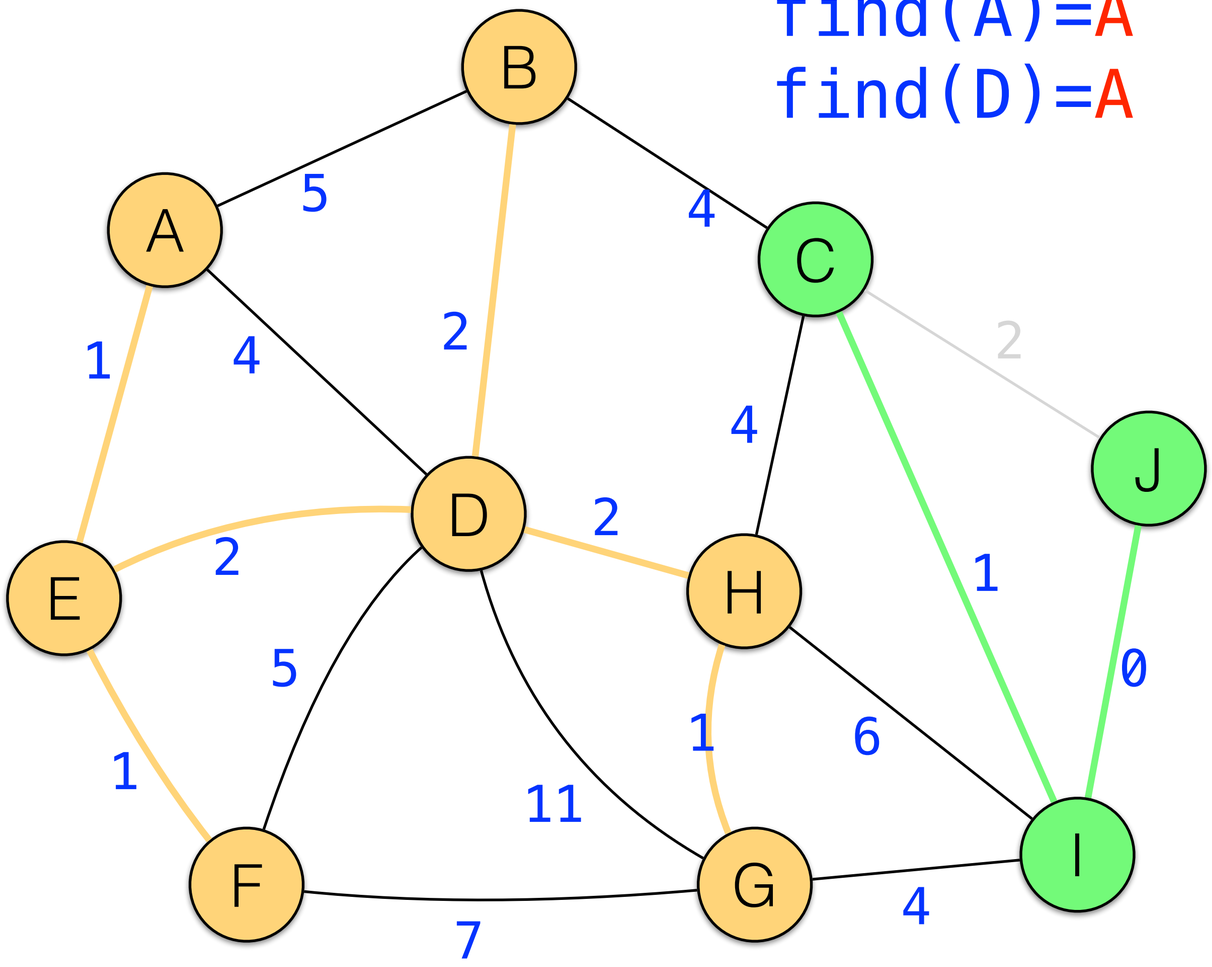



```
find(A)=A
find(D)=A
```

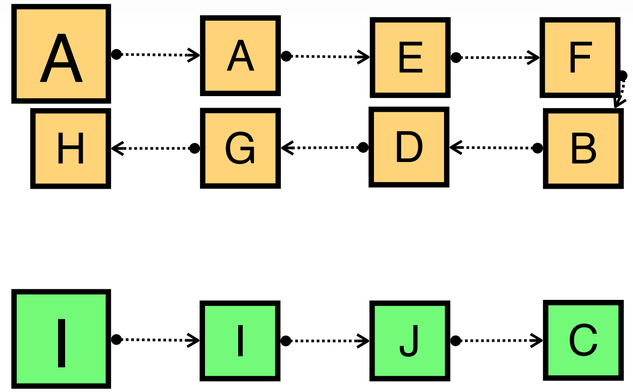


Kruskal MST

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



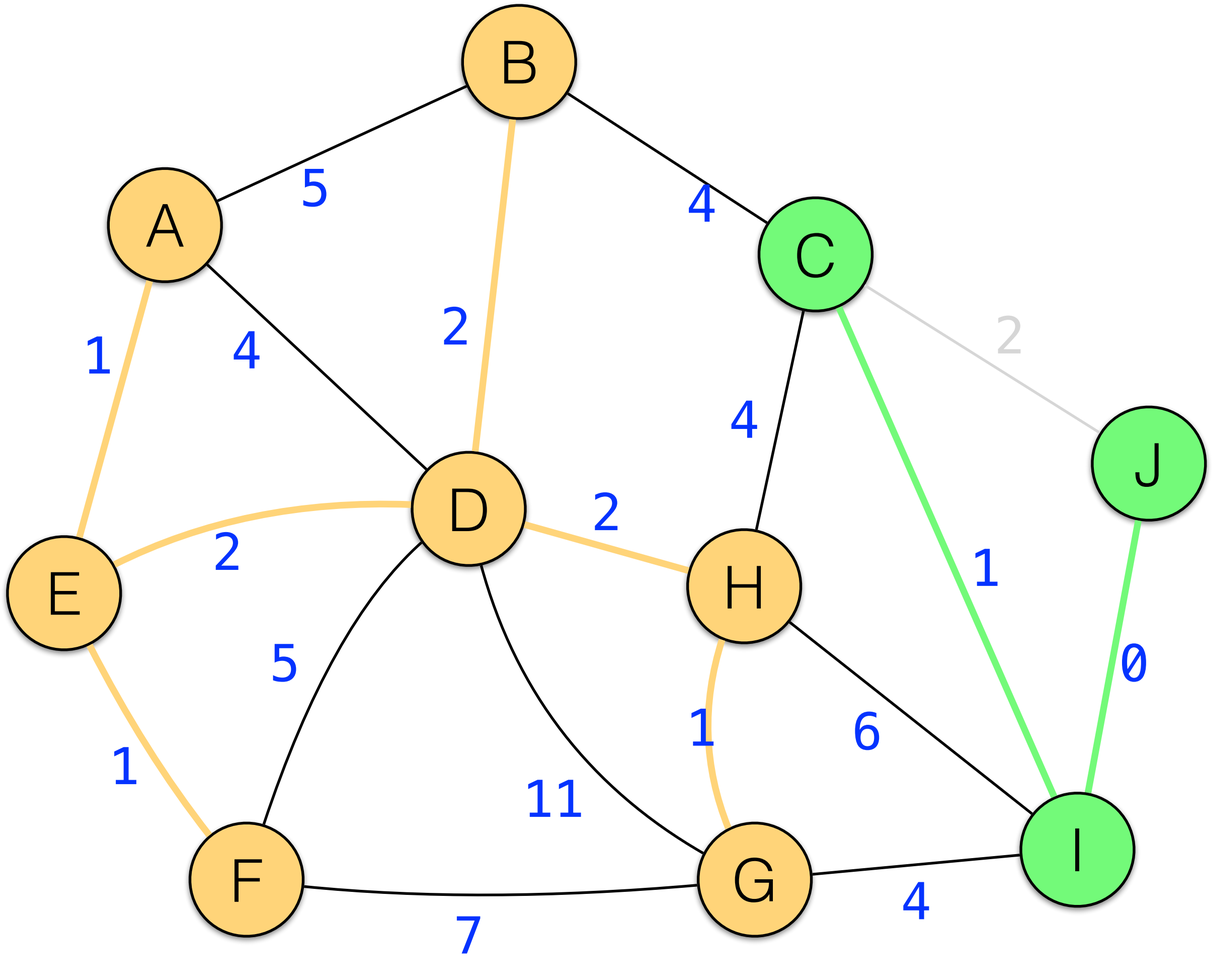

```
find(A)=A
find(D)=A
```

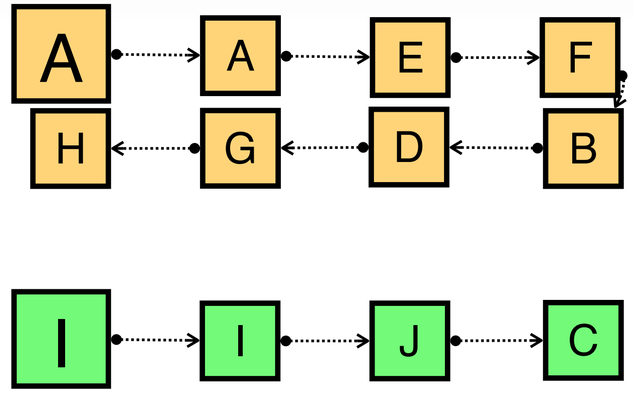


Do not add, it generates a cycle!

Kruskal MST

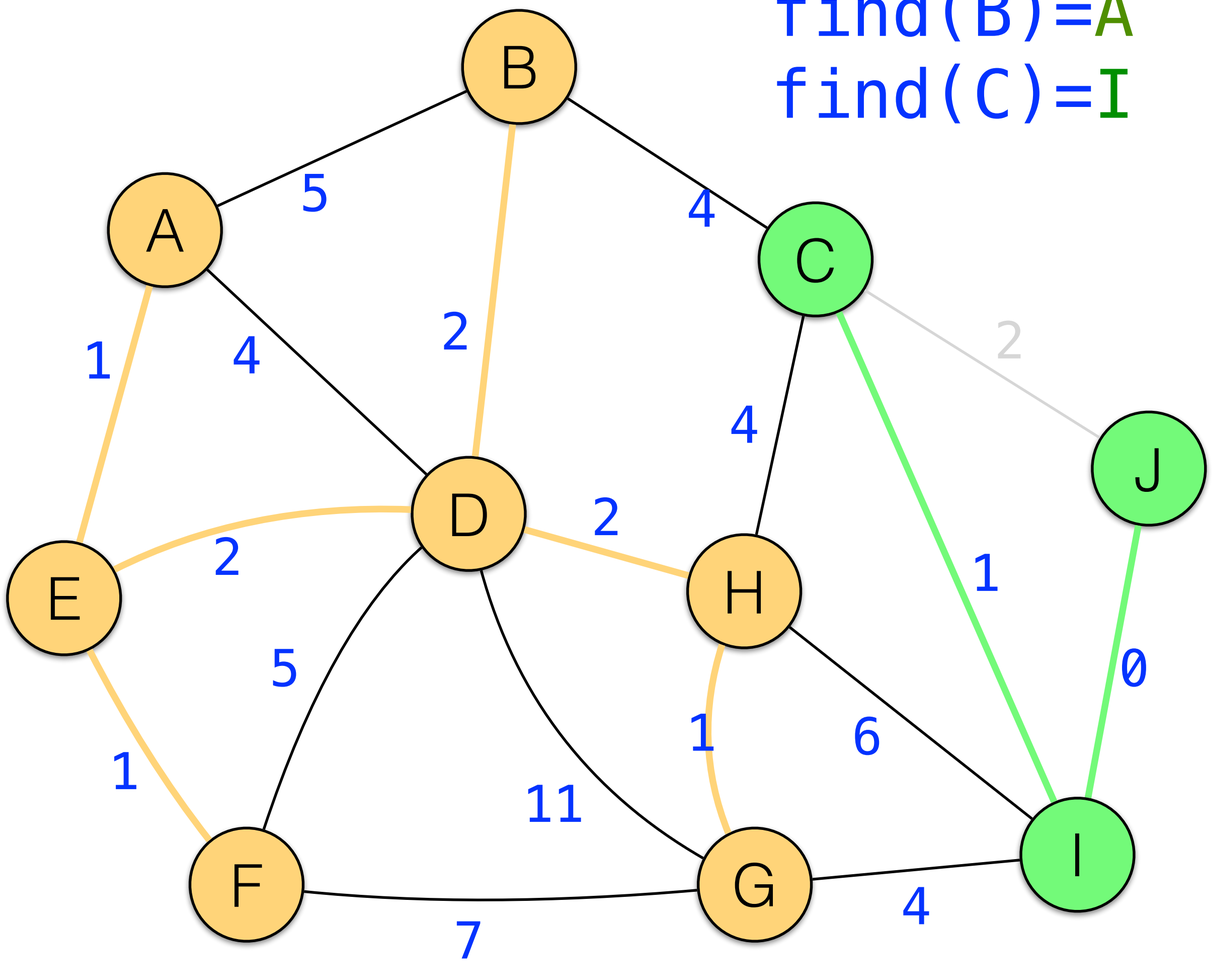
	I-J	A-E	C-I	E-F	G-H	B-D	C-J	D-E	D-H	A-D	B-C	C-H	G-I	A-B	D-F	H-I	F-G	D-G
d	0	1	1	1	1	2	2	2	2	4	4	4	4	5	5	6	7	11

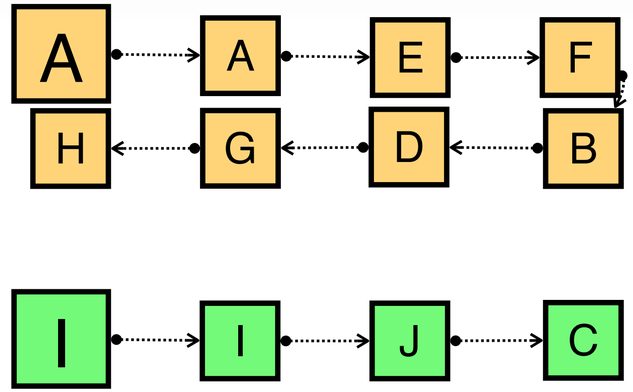




Kruskal MST

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

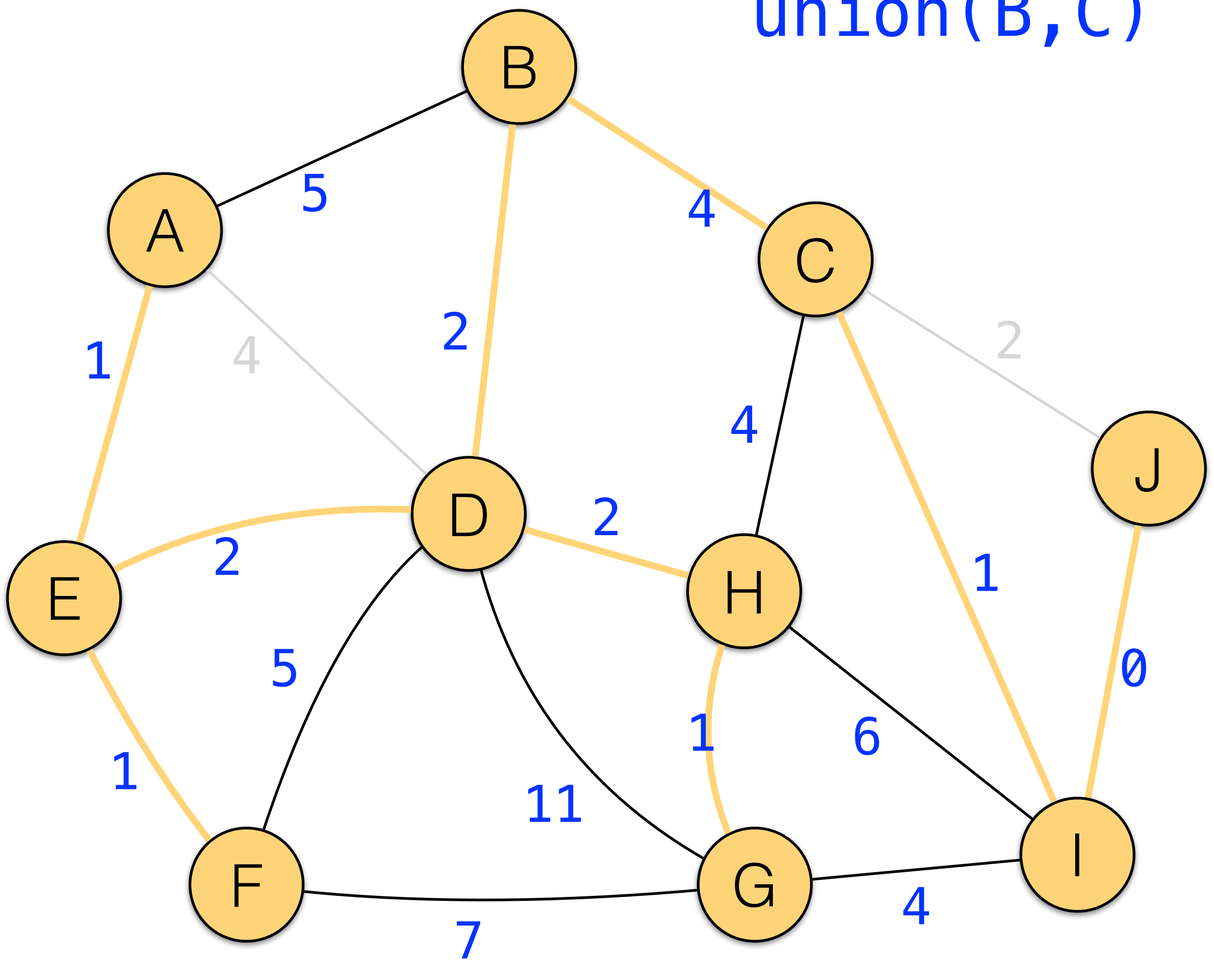


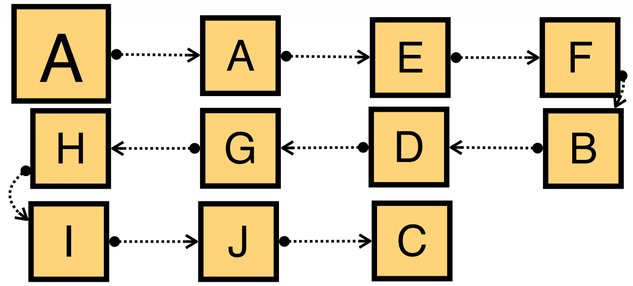


Kruskal MST

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

union(B,C)

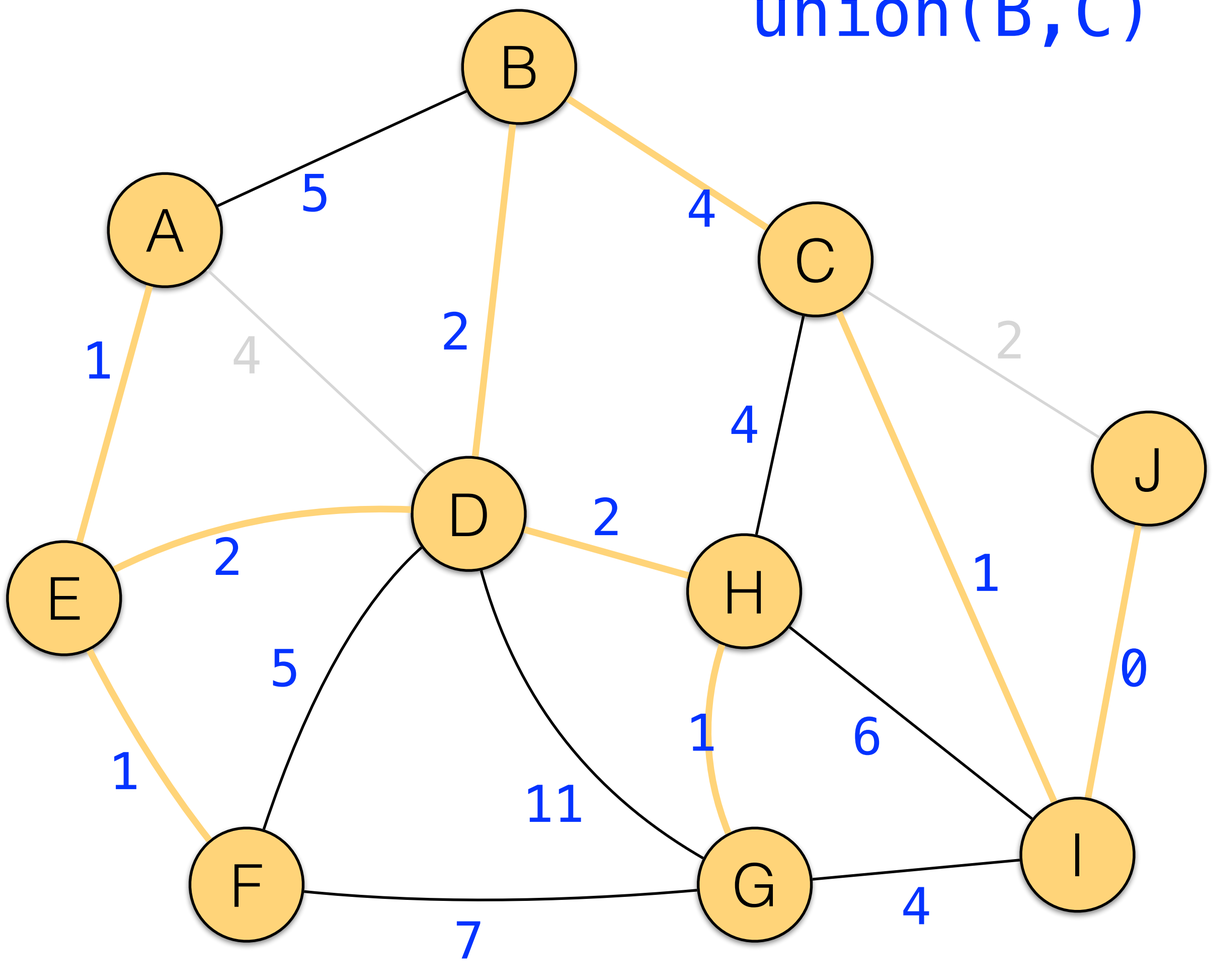


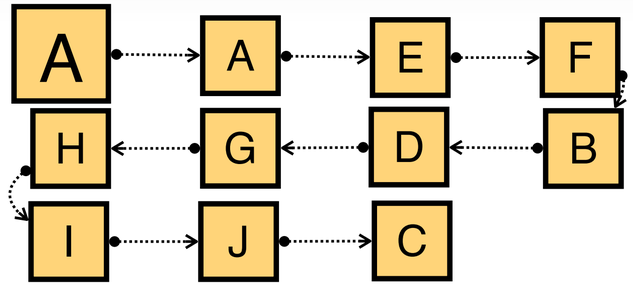


Kruskal MST

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

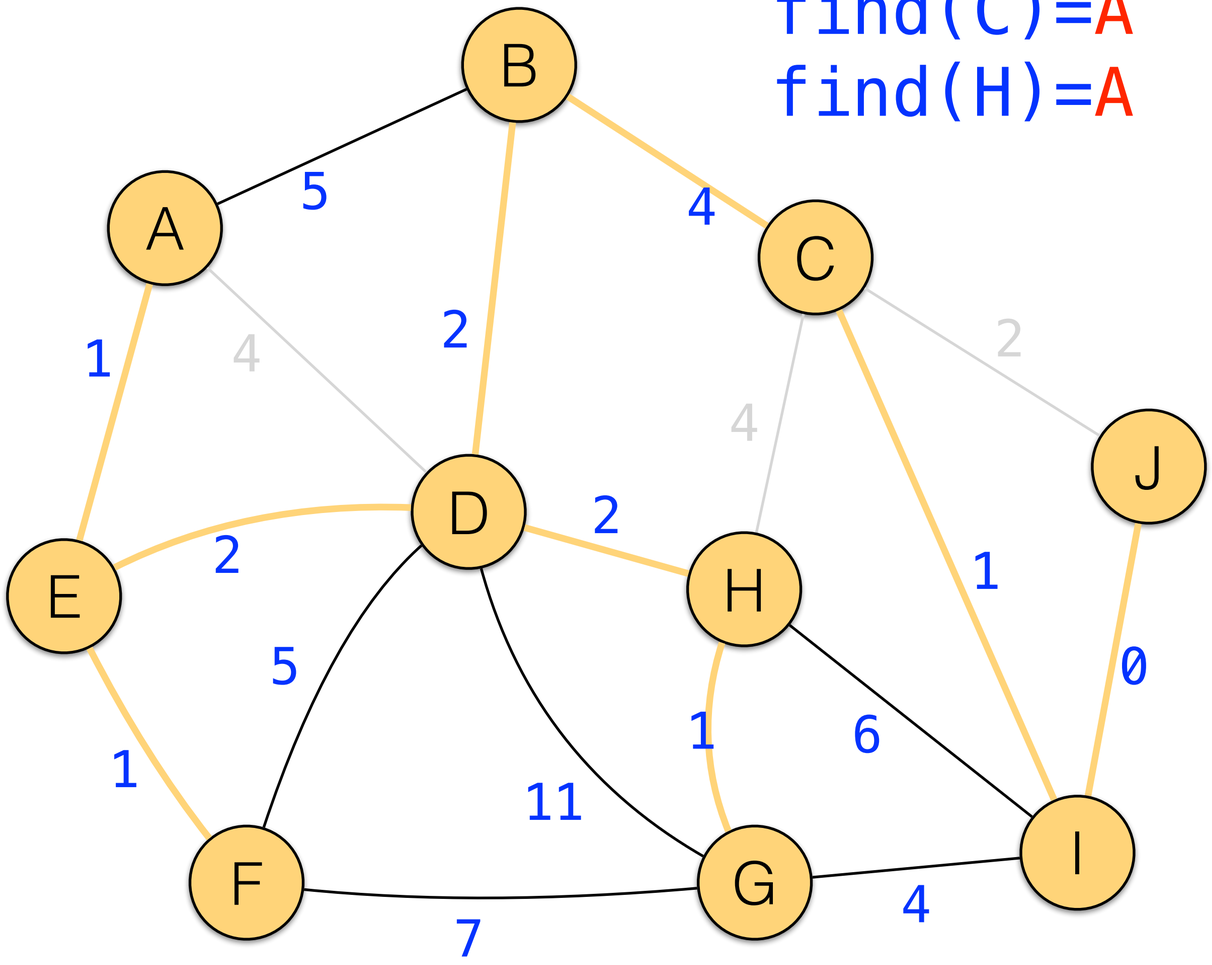
union(B,C)

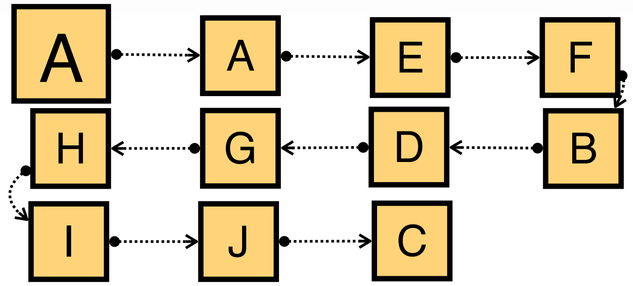




Kruskal MST

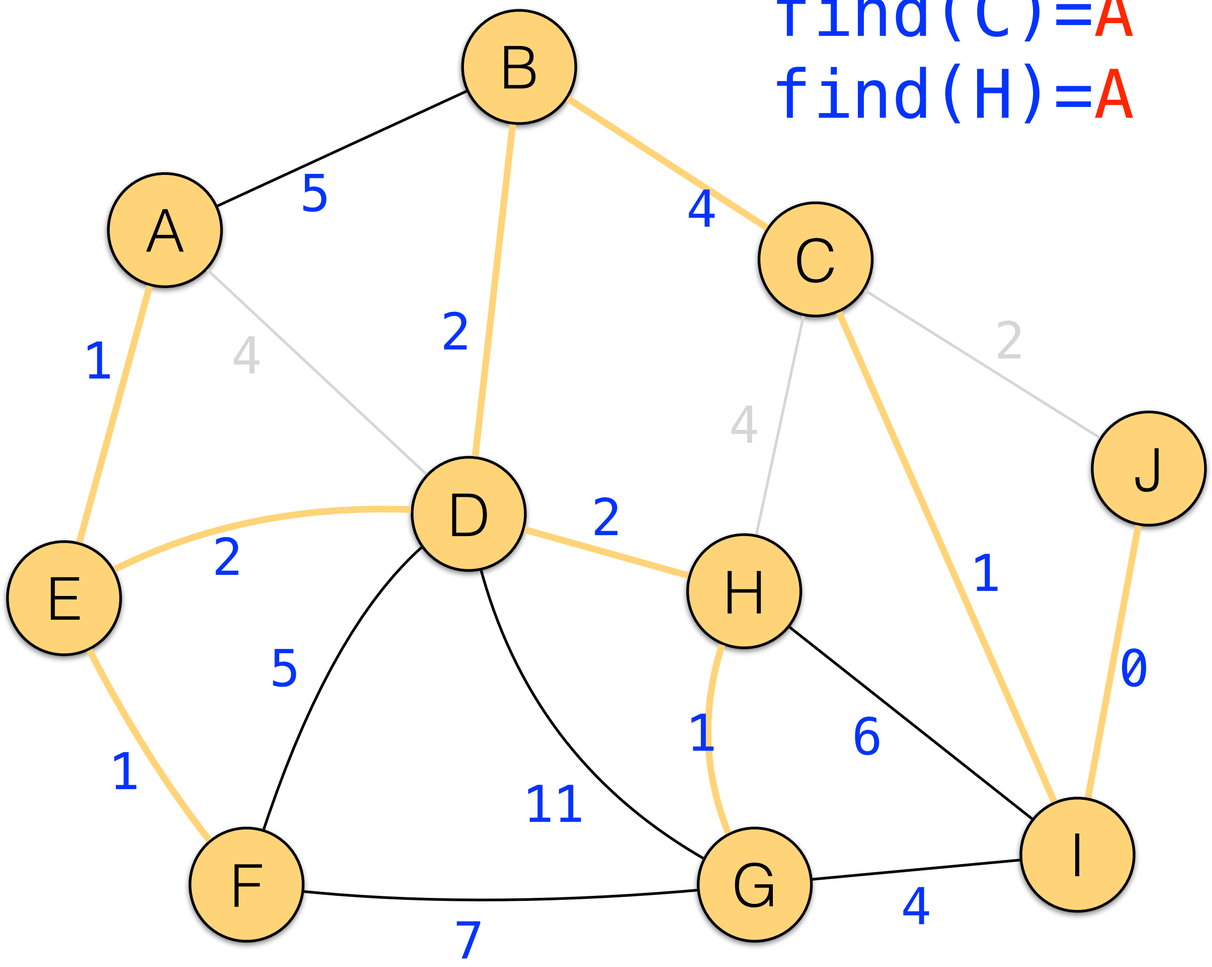
	I-J	A-E	C-I	E-F	G-H	B-D	C-J	D-E	D-H	A-D	B-C	C-H	G-I	A-B	D-F	H-I	F-G	D-G
d	0	1	1	1	1	2	2	2	2	4	4	4	4	5	5	6	7	11



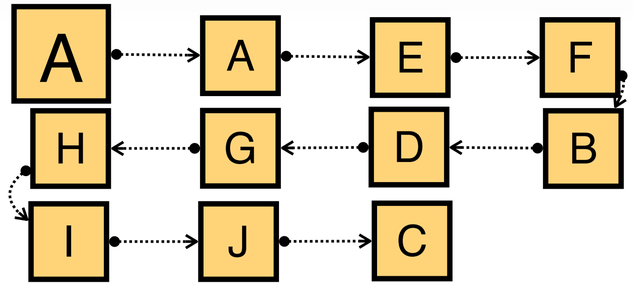


Kruskal MST

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



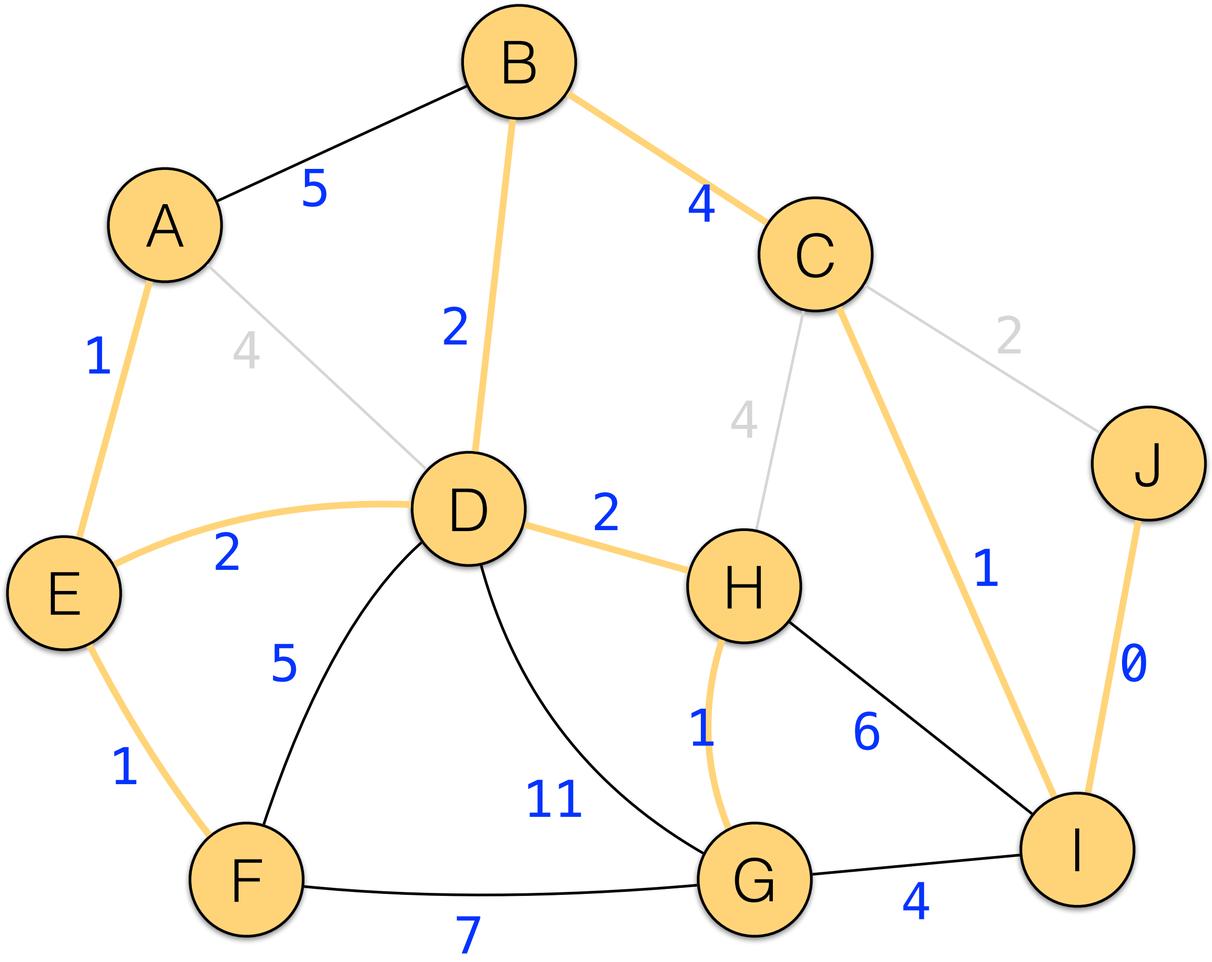
find(C)=A
find(H)=A

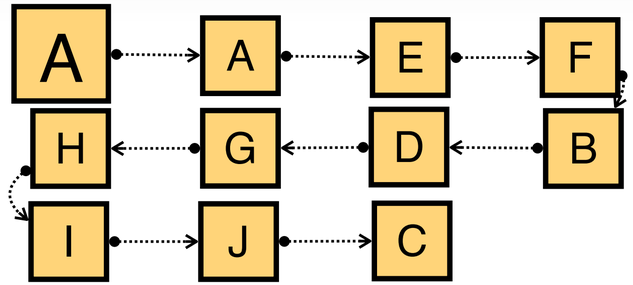


Do not add, it generates a cycle!

Kruskal MST

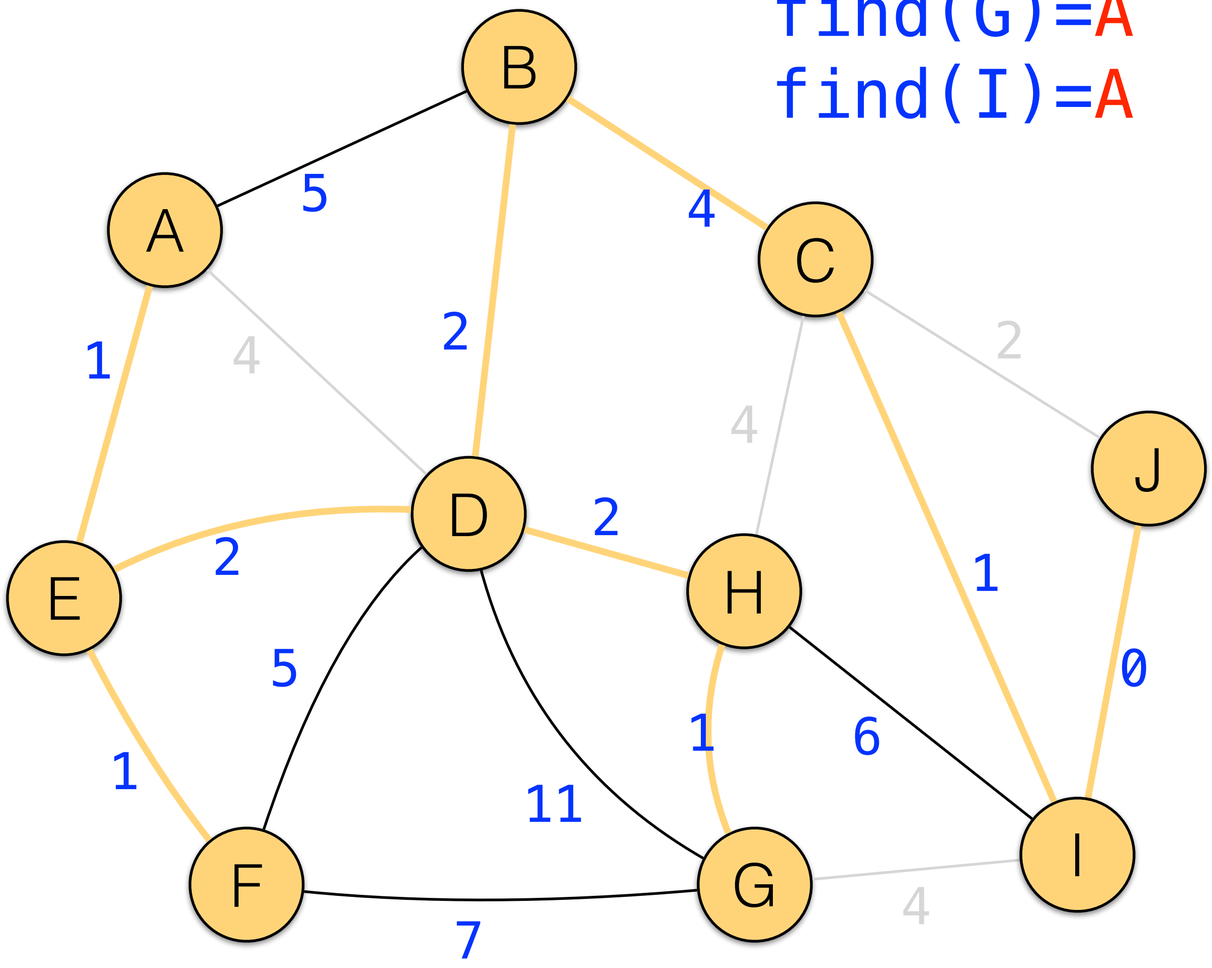
	I-J	A-E	C-I	E-F	G-H	B-D	C-J	D-E	D-H	A-D	B-C	C-H	G-I	A-B	D-F	H-I	F-G	D-G
d	0	1	1	1	1	2	2	2	2	4	4	4	4	5	5	6	7	11

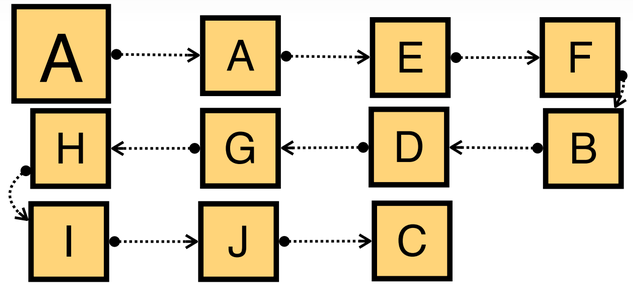




Kruskal MST

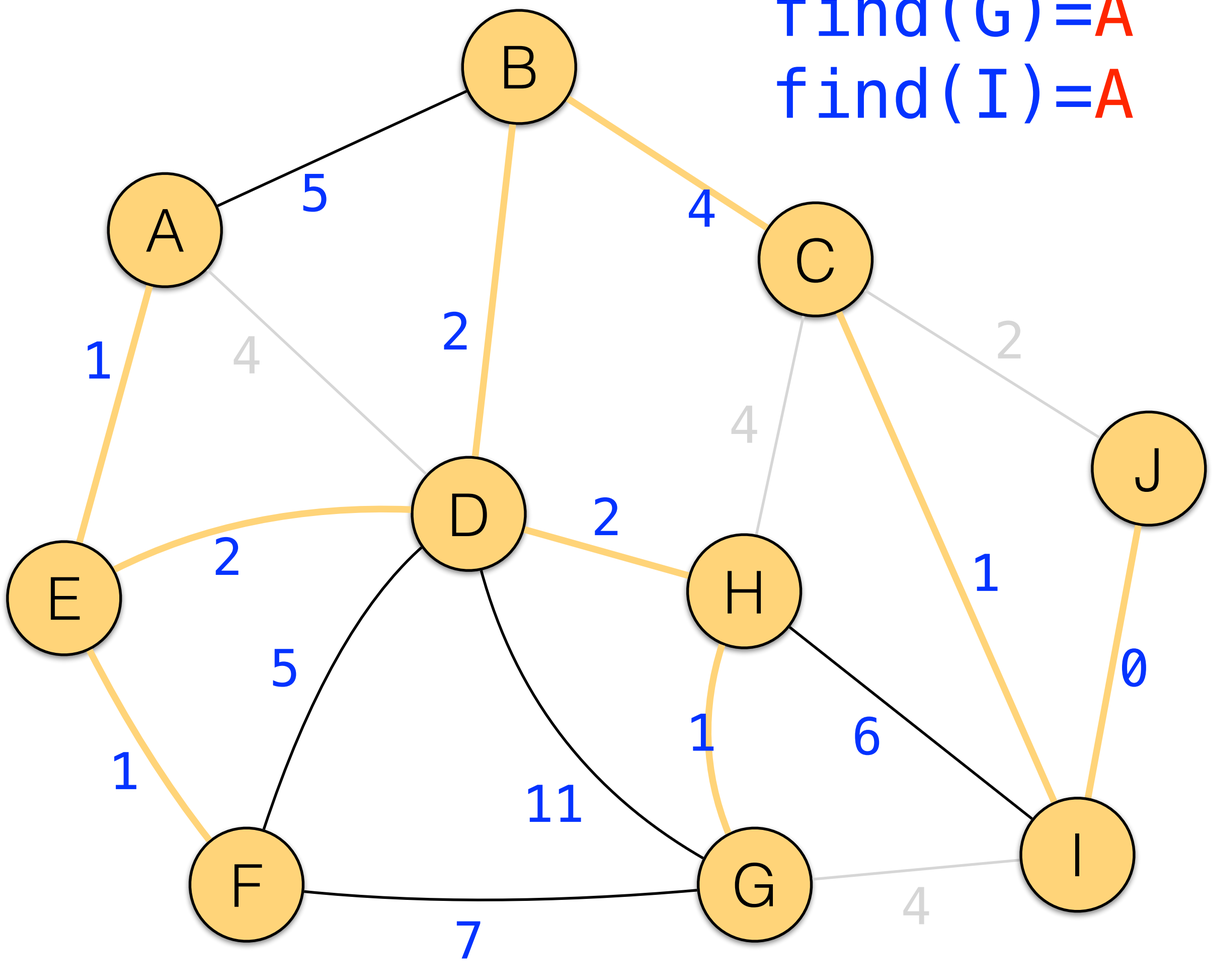
	I-J	A-E	C-I	E-F	G-H	B-D	C-J	D-E	D-H	A-D	B-C	C-H	G-I	A-B	D-F	H-I	F-G	D-G
d	0	1	1	1	1	2	2	2	2	4	4	4	4	5	5	6	7	11



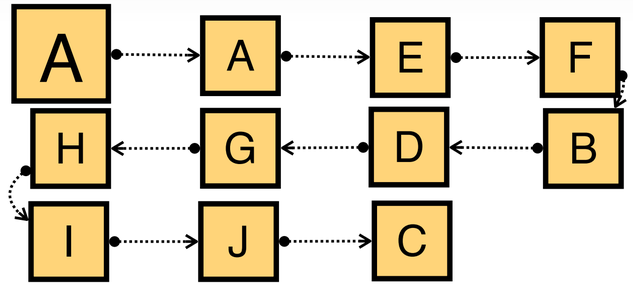


Kruskal MST

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



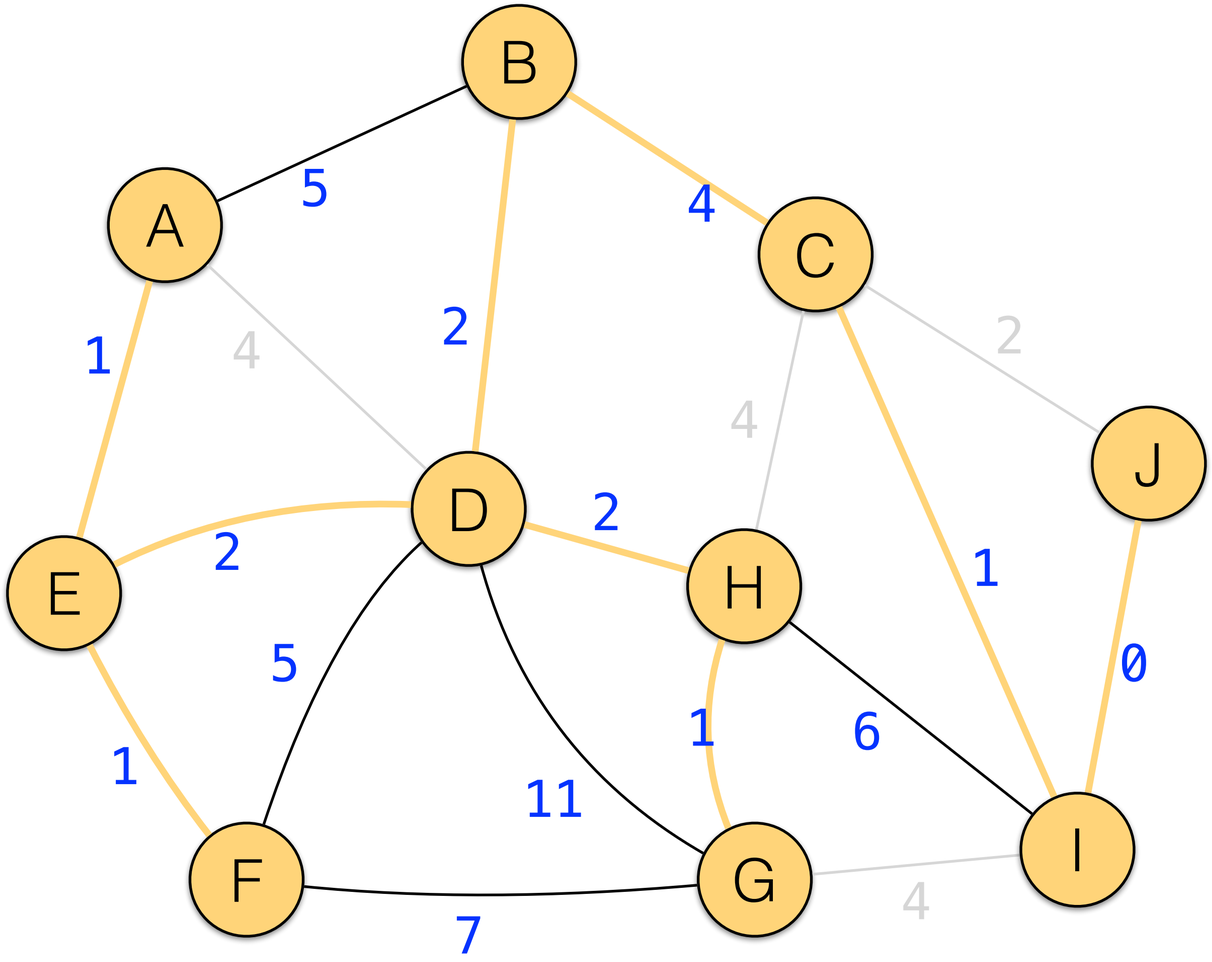
find(G)=A
find(I)=A

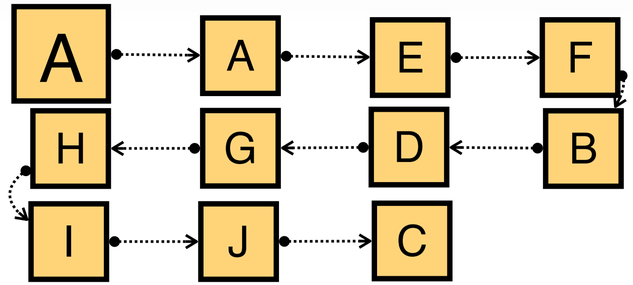


Do not add, it generates a cycle!

Kruskal MST

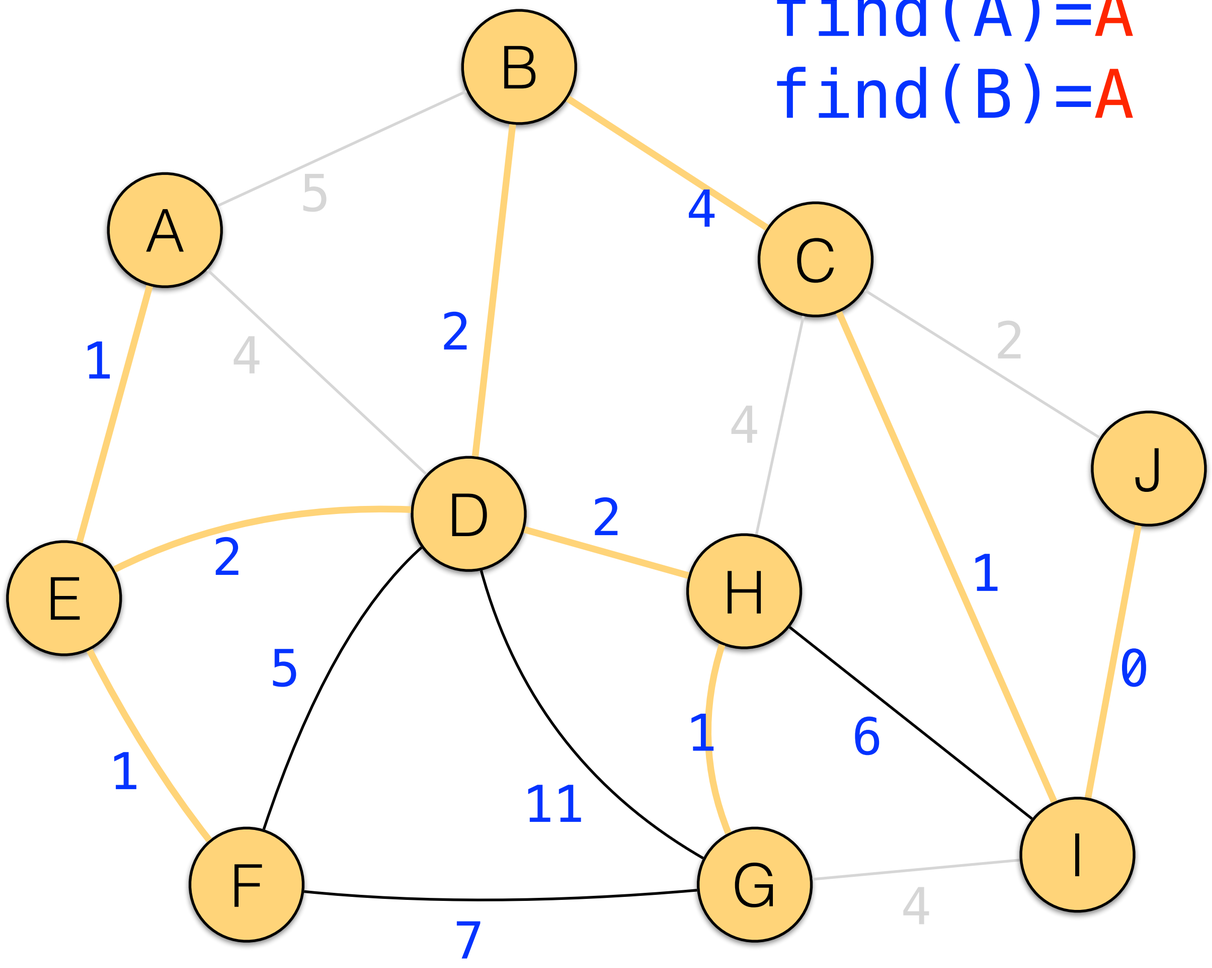
	I-J	A-E	C-I	E-F	G-H	B-D	C-J	D-E	D-H	A-D	B-C	C-H	G-I	A-B	D-F	H-I	F-G	D-G
d	0	1	1	1	1	2	2	2	2	4	4	4	4	5	5	6	7	11

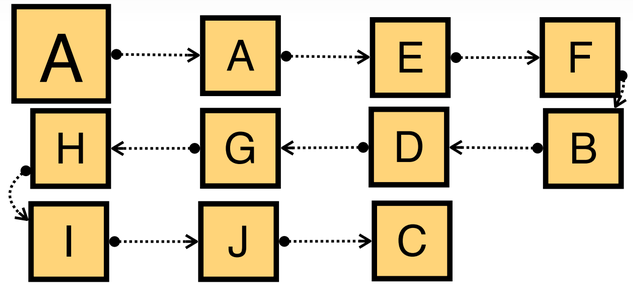




Kruskal MST

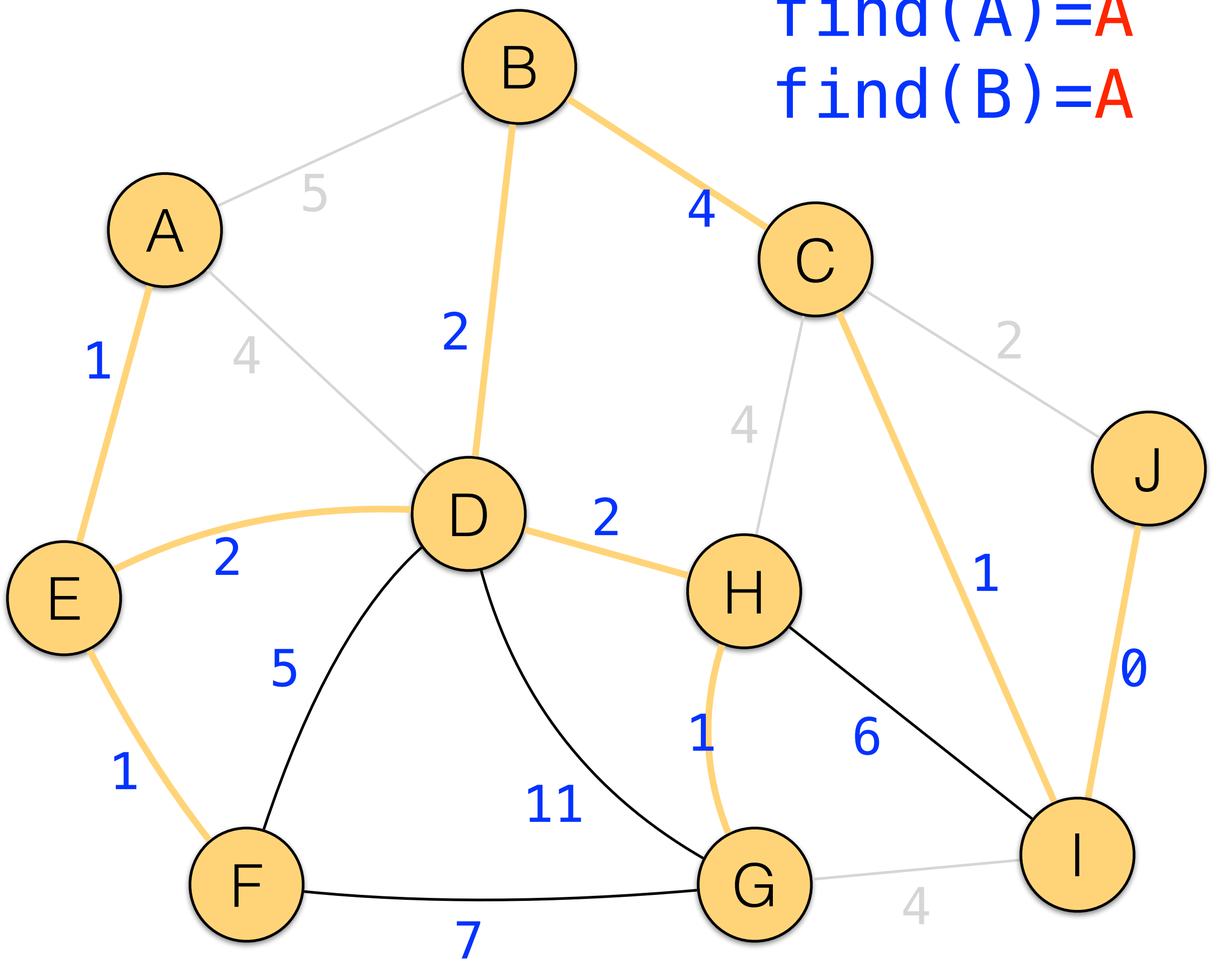
	I-J	A-E	C-I	E-F	G-H	B-D	C-J	D-E	D-H	A-D	B-C	C-H	G-I	A-B	D-F	H-I	F-G	D-G
d	0	1	1	1	1	2	2	2	2	4	4	4	4	5	5	6	7	11



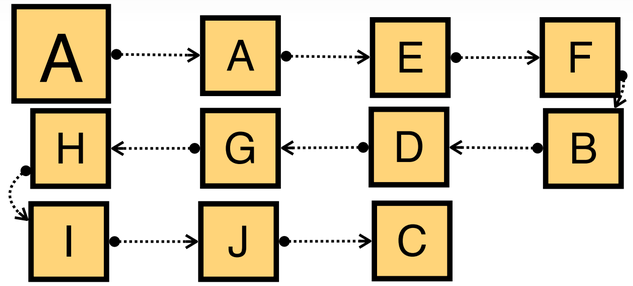


Kruskal MST

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



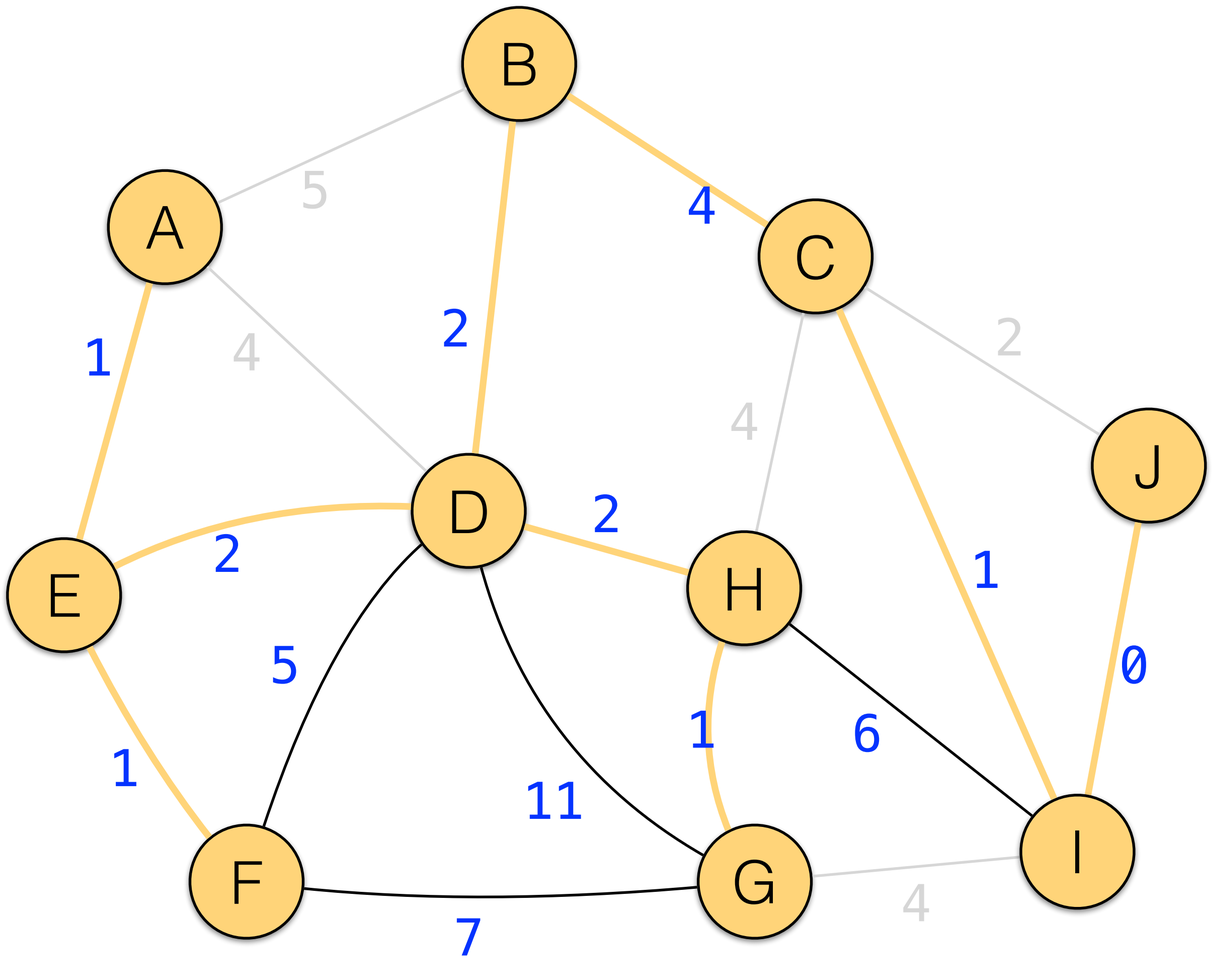

```
find(A)=A
find(B)=A
```

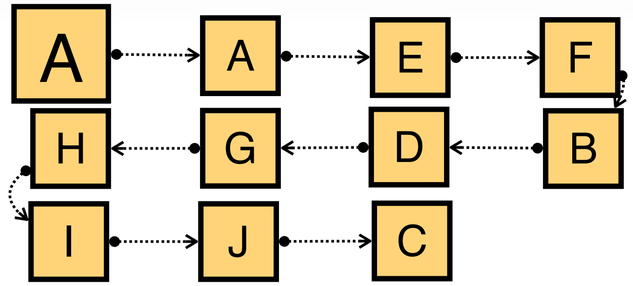


Do not add, it generates a cycle!

Kruskal MST

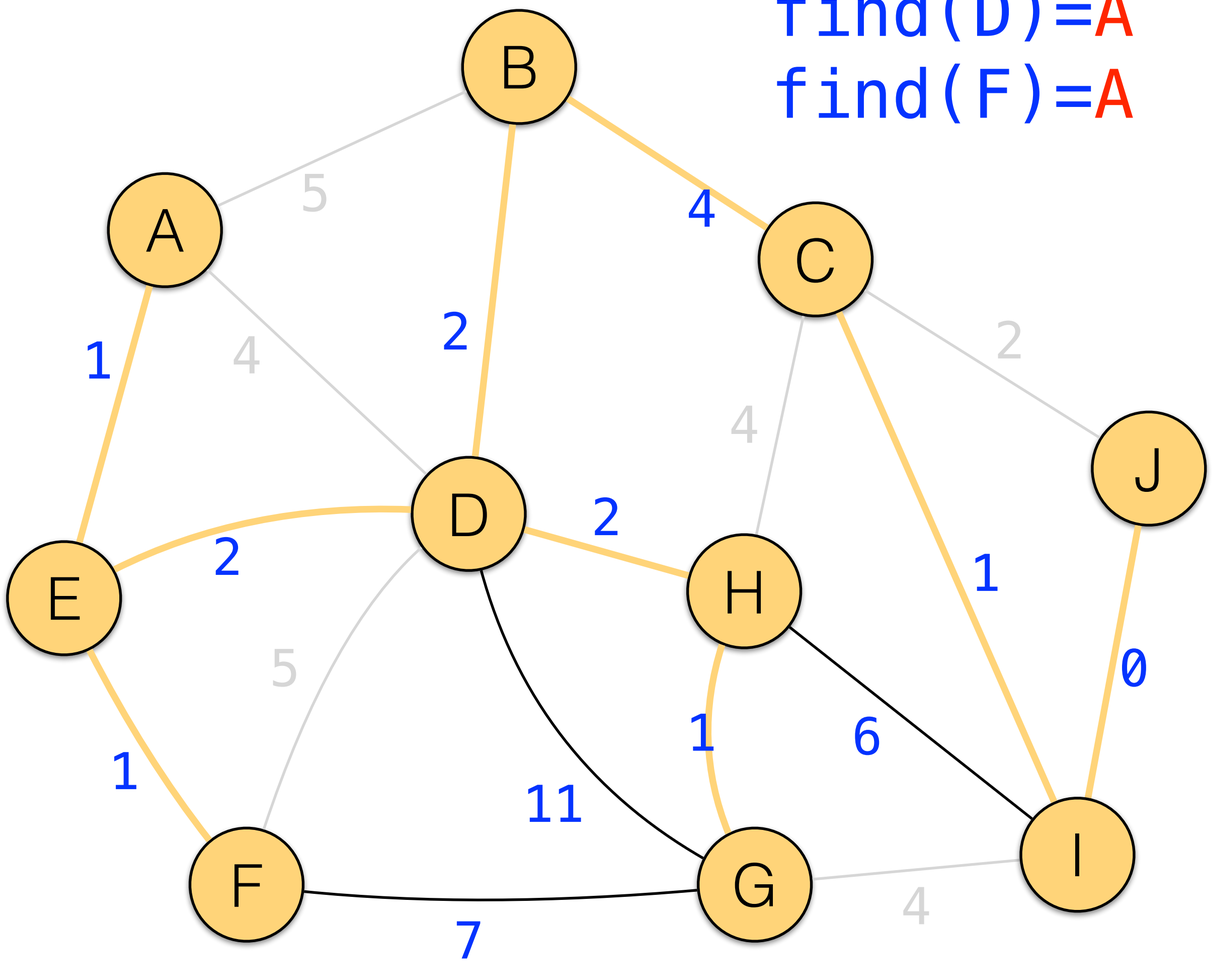
	I-J	A-E	C-I	E-F	G-H	B-D	C-J	D-E	D-H	A-D	B-C	C-H	G-I	A-B	D-F	H-I	F-G	D-G
d	0	1	1	1	1	2	2	2	2	4	4	4	4	5	5	6	7	11

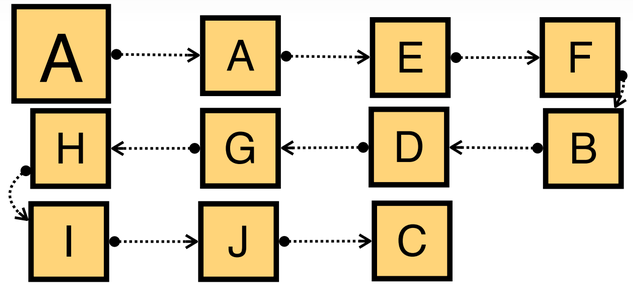




Kruskal MST

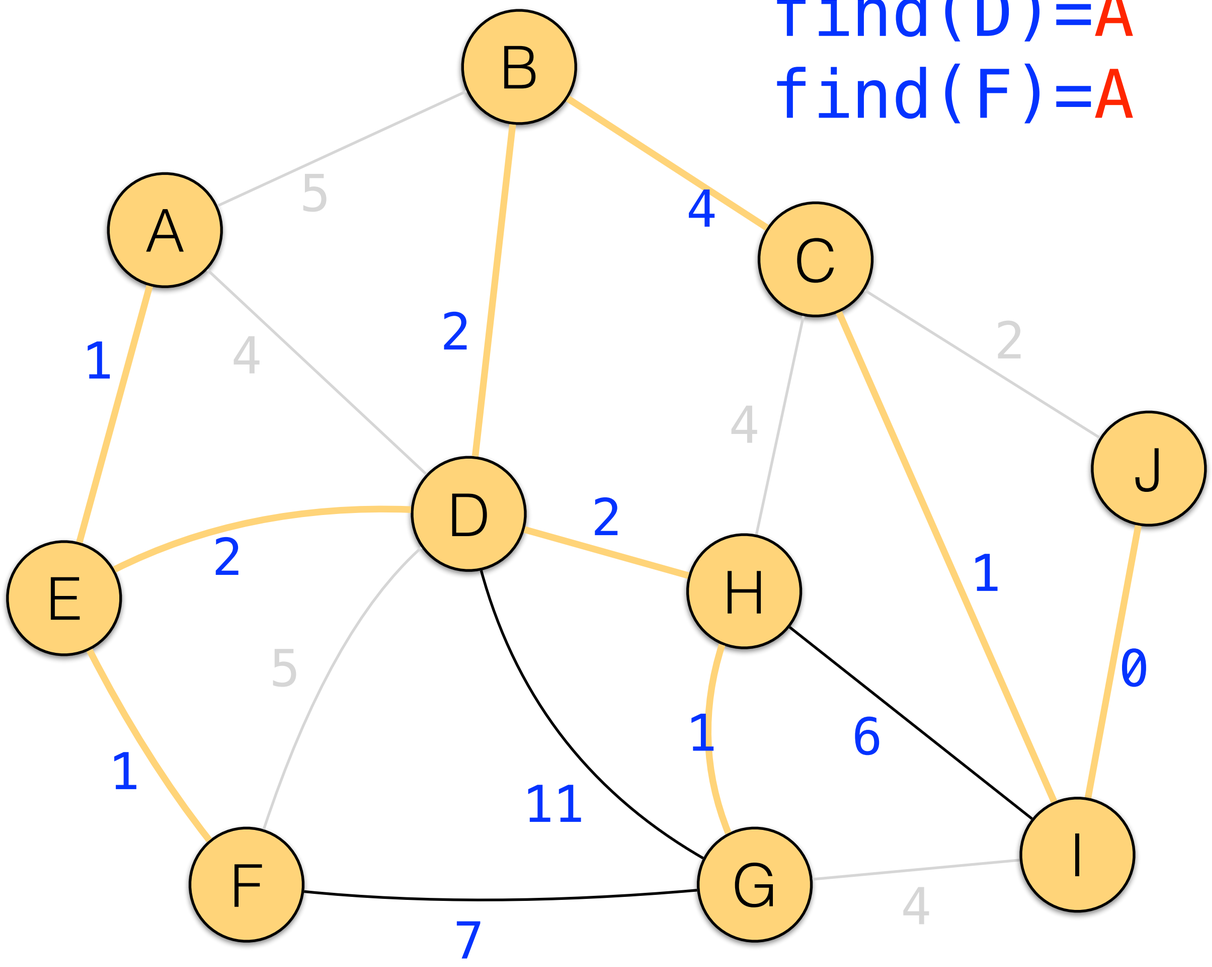
	I-J	A-E	C-I	E-F	G-H	B-D	C-J	D-E	D-H	A-D	B-C	C-H	G-I	A-B	D-F	H-I	F-G	D-G
d	0	1	1	1	1	2	2	2	2	4	4	4	4	5	5	6	7	11



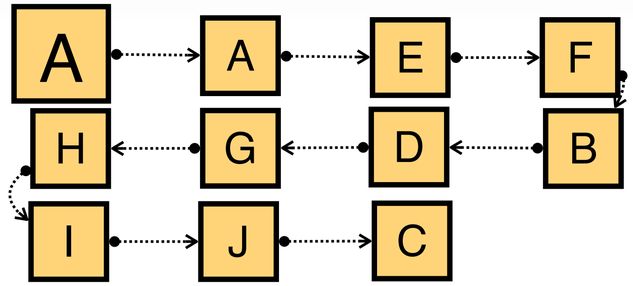


Kruskal MST

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



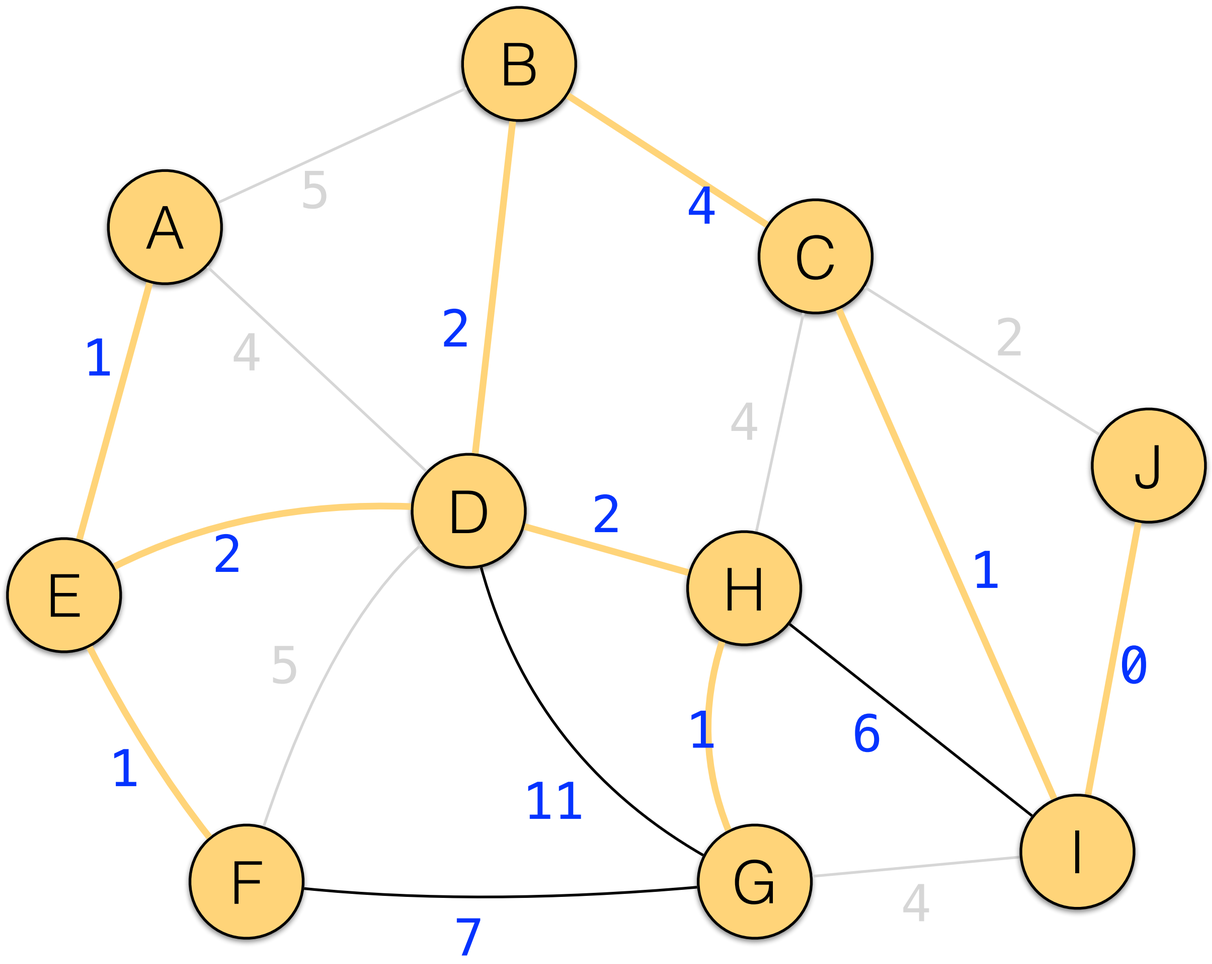
find(D)=A
find(F)=A

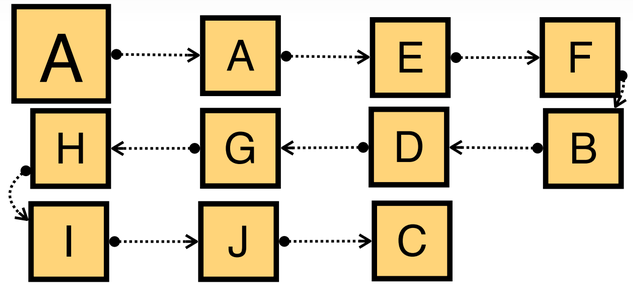


Do not add, it generates a cycle!

Kruskal MST

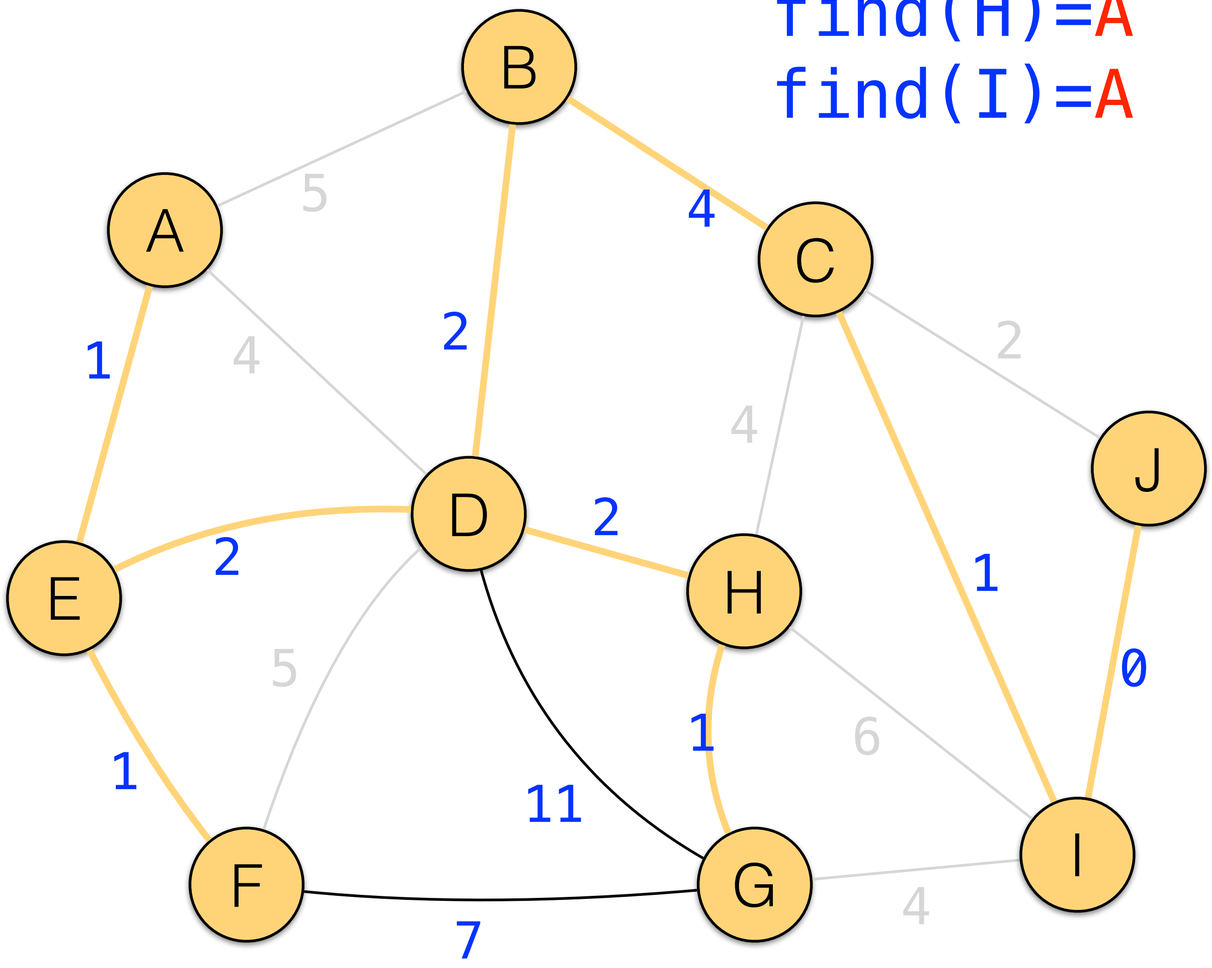
	I-J	A-E	C-I	E-F	G-H	B-D	C-J	D-E	D-H	A-D	B-C	C-H	G-I	A-B	D-F	H-I	F-G	D-G
d	0	1	1	1	1	2	2	2	2	4	4	4	4	5	5	6	7	11

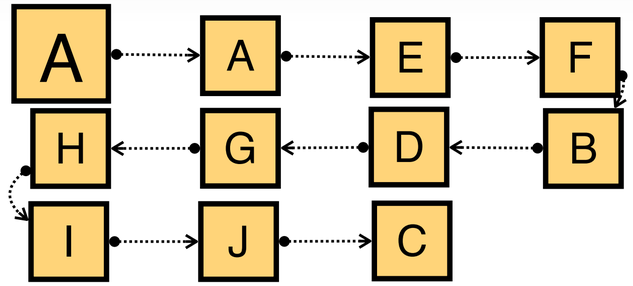




Kruskal MST

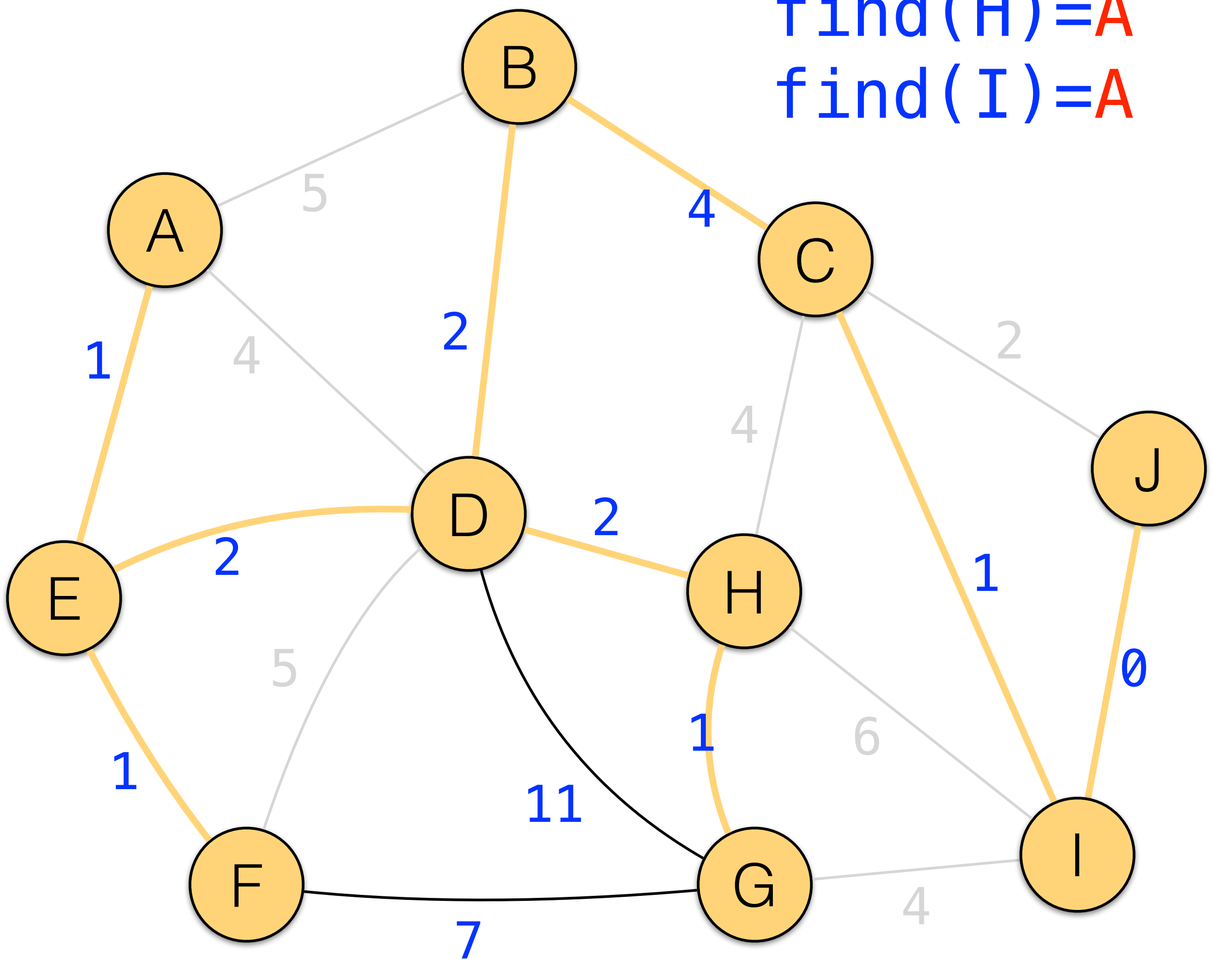
	I-J	A-E	C-I	E-F	G-H	B-D	C-J	D-E	D-H	A-D	B-C	C-H	G-I	A-B	D-F	H-I	F-G	D-G
d	0	1	1	1	1	2	2	2	2	4	4	4	4	5	5	6	7	11



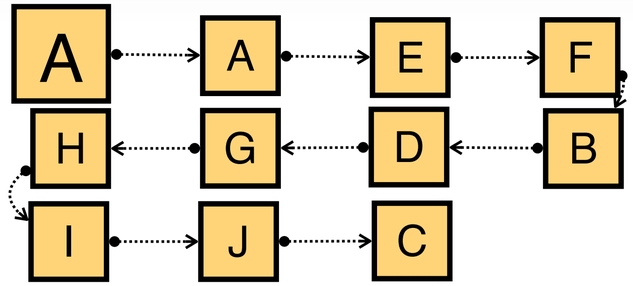


Kruskal MST

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



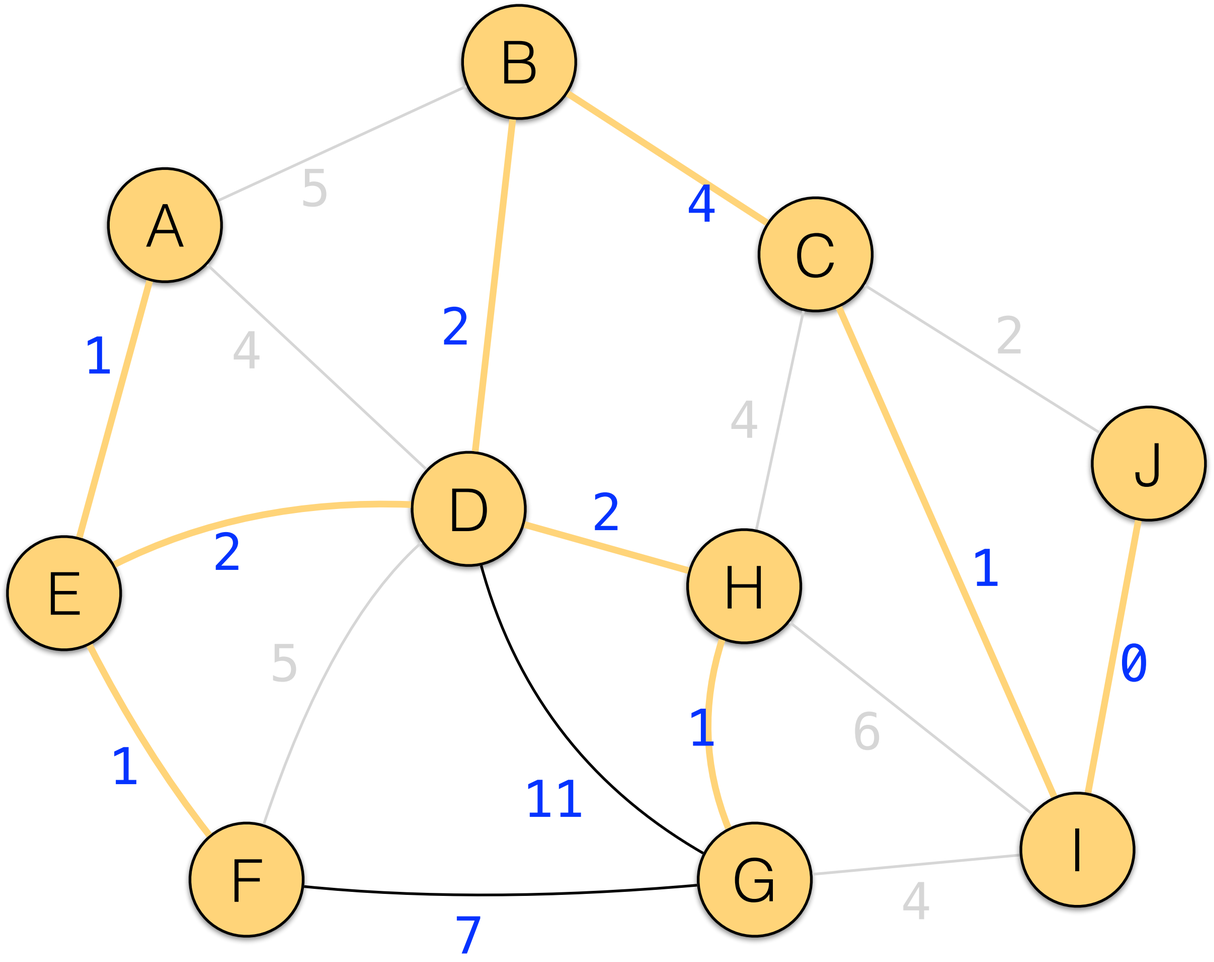
find(H)=A
find(I)=A

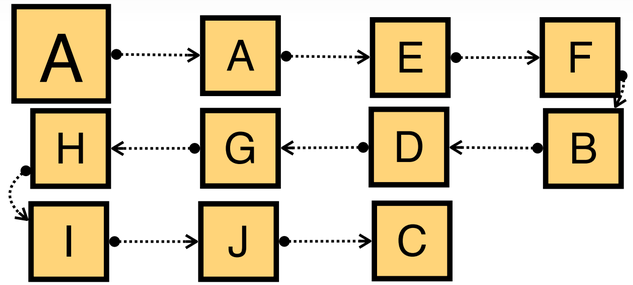


Do not add, it generates a cycle!

Kruskal MST

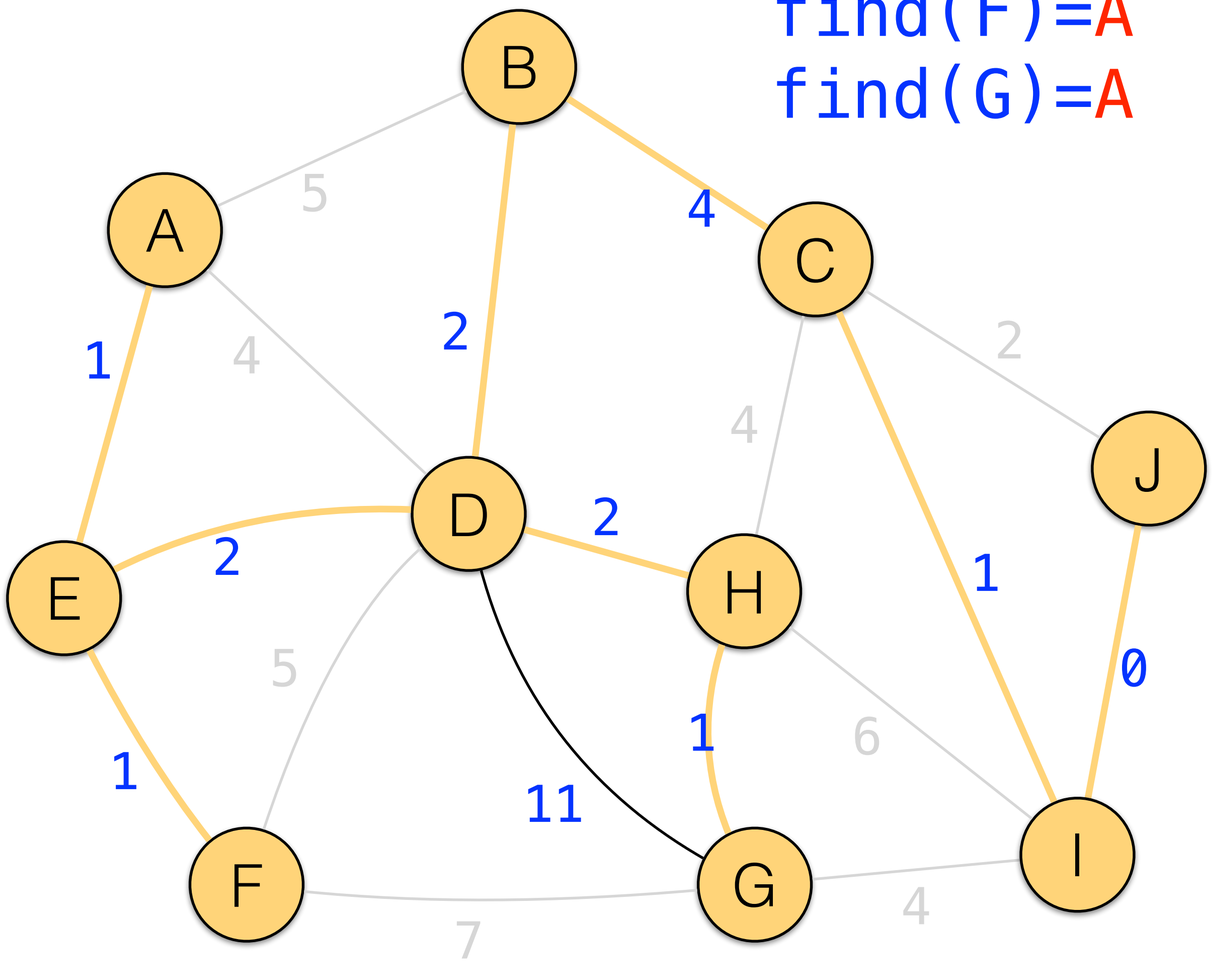
	I-J	A-E	C-I	E-F	G-H	B-D	C-J	D-E	D-H	A-D	B-C	C-H	G-I	A-B	D-F	H-I	F-G	D-G
d	0	1	1	1	1	2	2	2	2	4	4	4	4	5	5	6	7	11

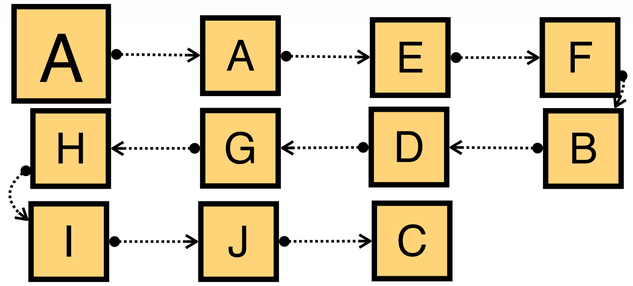




Kruskal MST

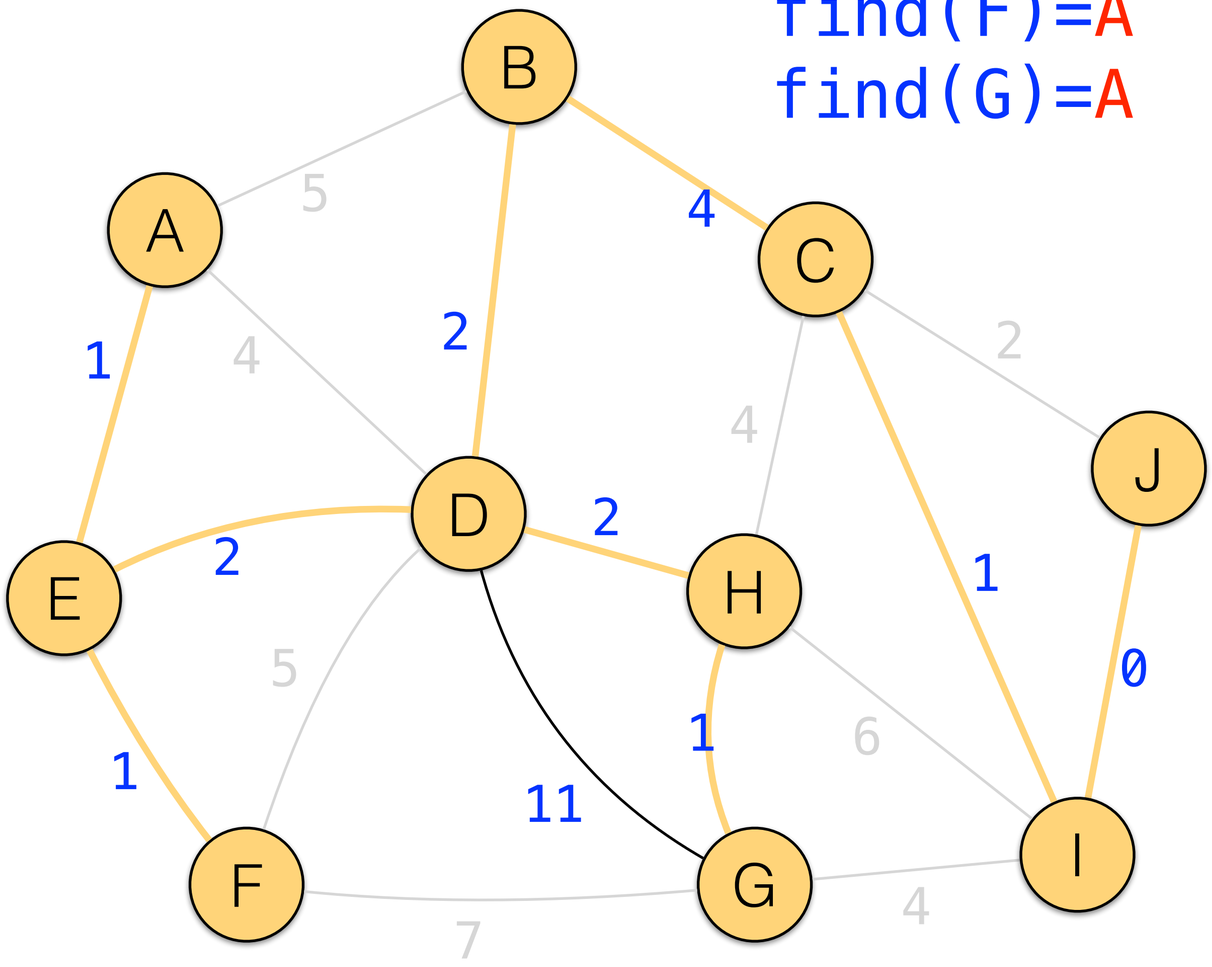
	I-J	A-E	C-I	E-F	G-H	B-D	C-J	D-E	D-H	A-D	B-C	C-H	G-I	A-B	D-F	H-I	F-G	D-G
d	0	1	1	1	1	2	2	2	2	4	4	4	4	5	5	6	7	11



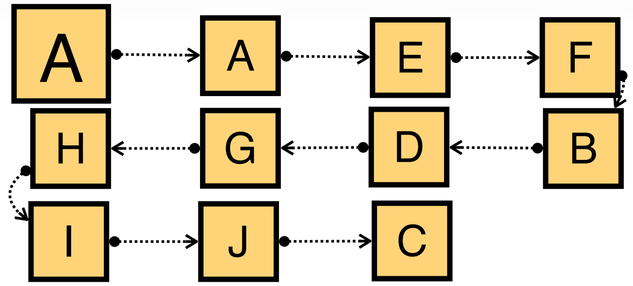


Kruskal MST

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



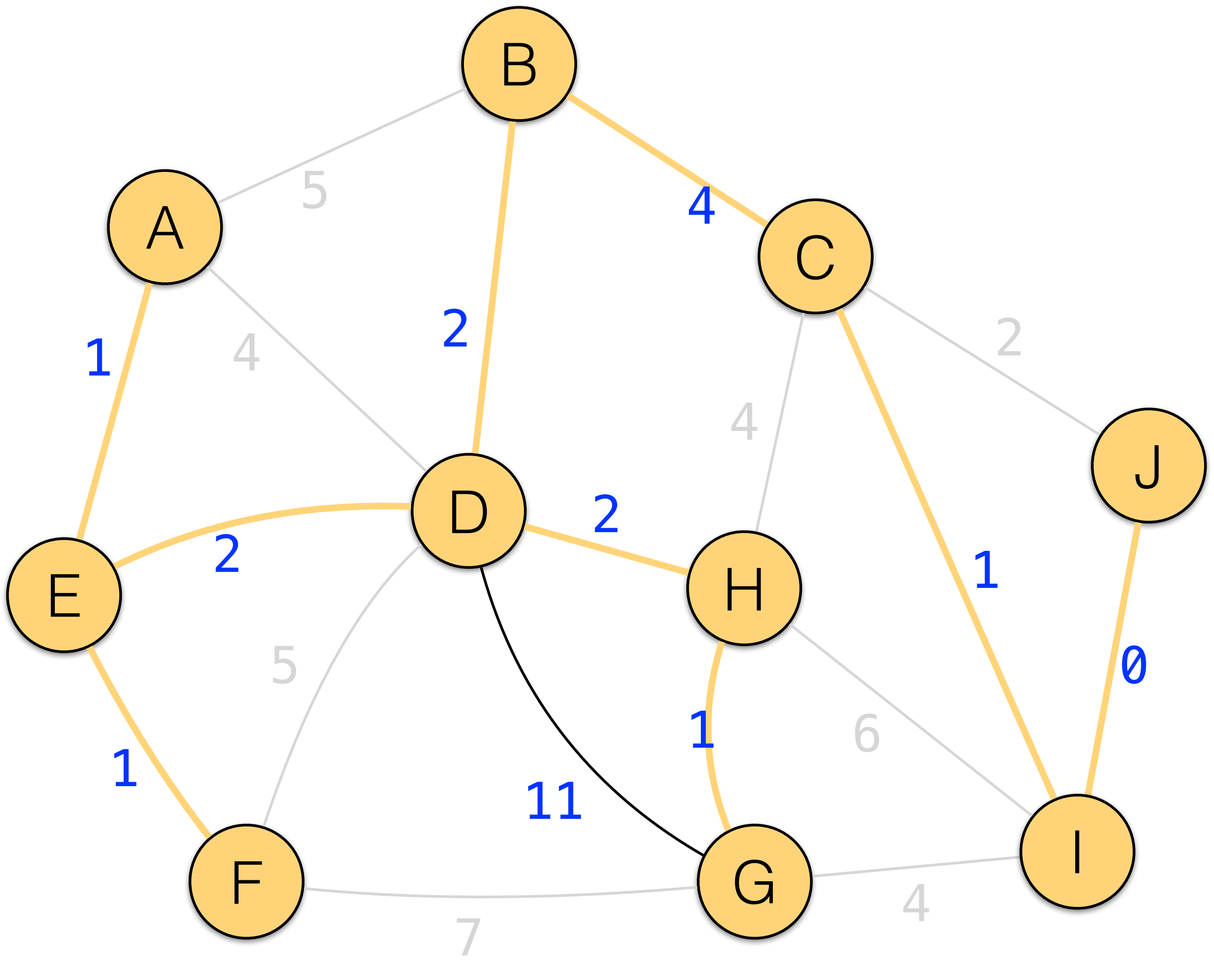
find(F)=A
find(G)=A

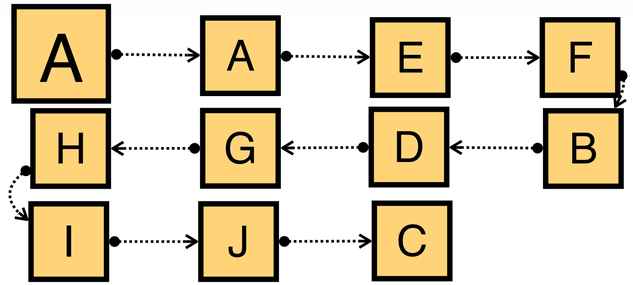


Do not add, it generates a cycle!

Kruskal MST

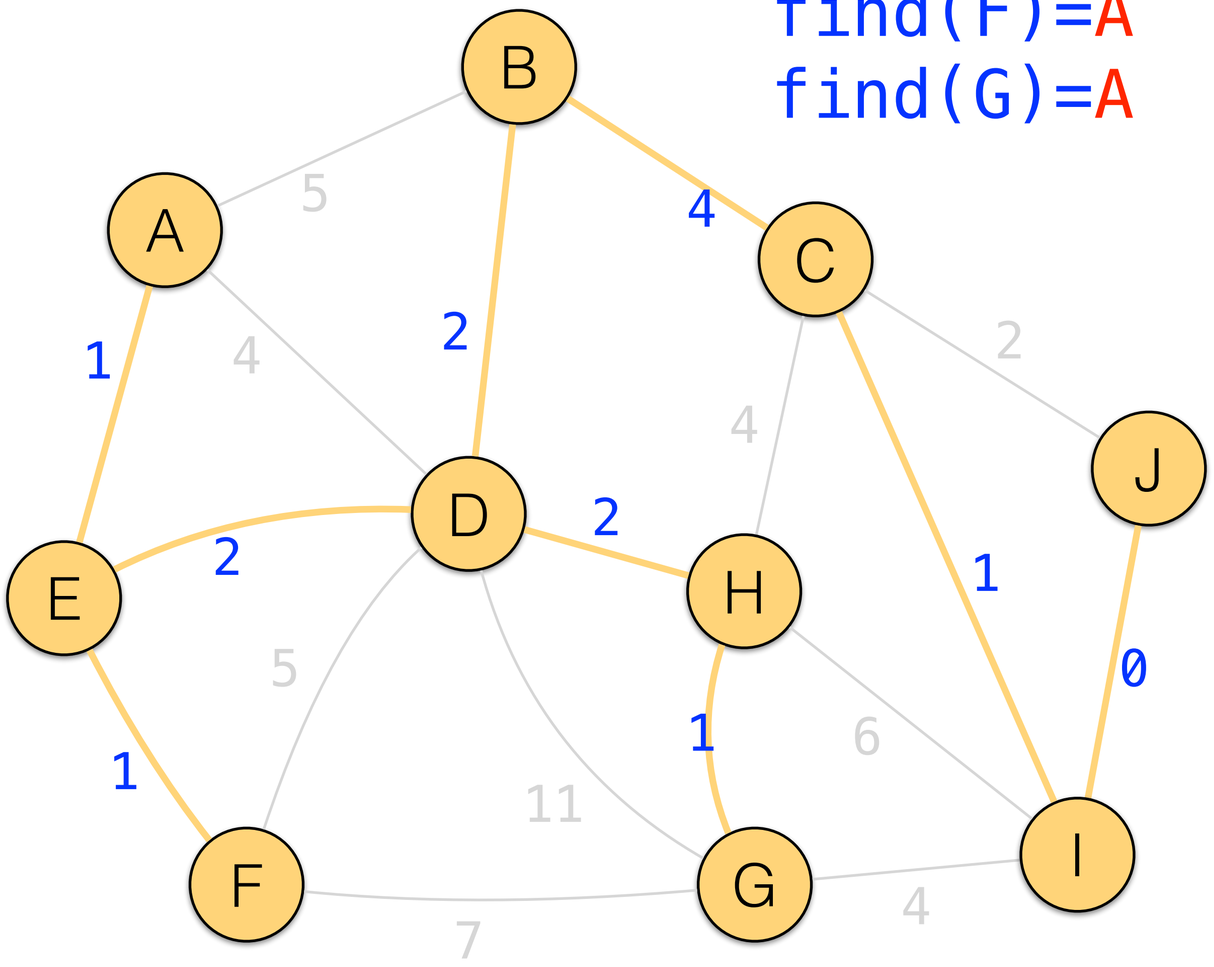
	I-J	A-E	C-I	E-F	G-H	B-D	C-J	D-E	D-H	A-D	B-C	C-H	G-I	A-B	D-F	H-I	F-G	D-G
d	0	1	1	1	1	2	2	2	2	4	4	4	4	5	5	6	7	11

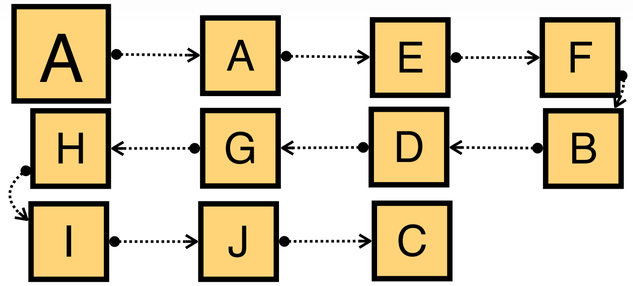




Kruskal MST

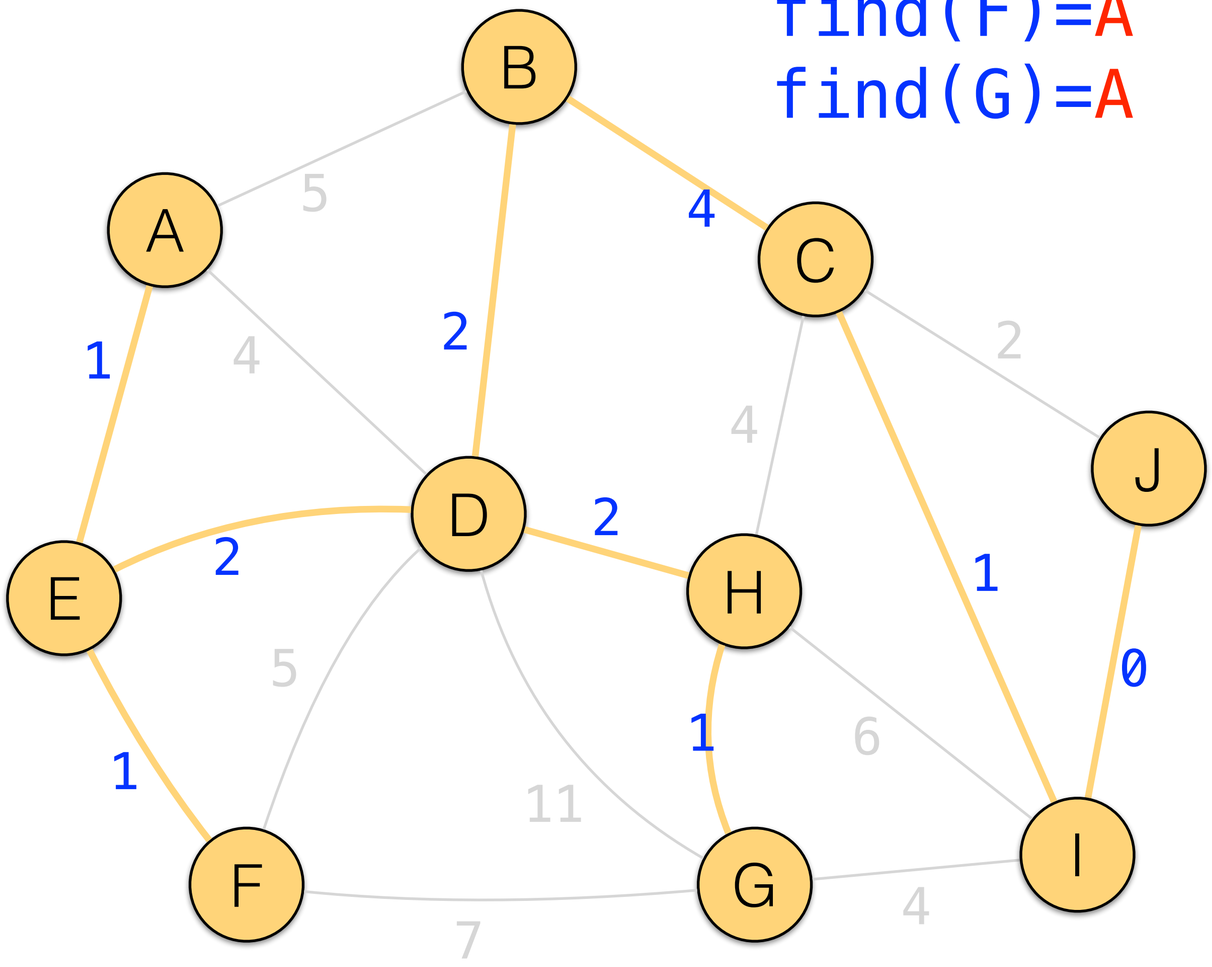
	I-J	A-E	C-I	E-F	G-H	B-D	C-J	D-E	D-H	A-D	B-C	C-H	G-I	A-B	D-F	H-I	F-G	D-G
d	0	1	1	1	1	2	2	2	2	4	4	4	4	5	5	6	7	11



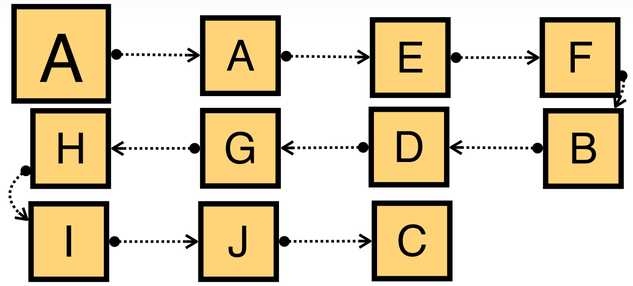


Kruskal MST

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



find(F)=A
find(G)=A



Do not add, it generates a cycle!

https://www.codechef.com/problems/ABROADS

In Ancient Berland, there were N towns, along with M bidirectional roads connecting them. With time, some roads became unusable, and nobody repaired them.
As a person who is fond of Ancient Berland history, you now want to undertake a small research study. For this purpose, you want to write a program capable of processing the following kinds of queries:

- D K : meaning that the road numbered K in the input became unusable. The road numbers are 1-indexed.
- P A x : meaning that the population of the A^{th} town became x.

Let's call a subset of towns a region if it is possible to get from each town in the subset to every other town in the subset by the usable (those, which haven't already been destroyed) roads, possibly, via some intermediary cities of this subset. The population of the region is, then, the sum of populations of all the towns in the region.
You are given the initial road system, the initial population in each town and Q queries, each being one of two types above. Your task is to maintain the size of the most populated region after each query.

https://www.codechef.com/problems/ABROADS

Input

The first line of each test case contains three space-separated integers — N, M, and Q — denoting the number of cities, the number of roads, and the number of queries, respectively.
The following line contains N space-separated integers, the i^{th} of which denotes the initial population of the i^{th} city.
The j^{th} of the following M lines contains a pair of space-separated integers — Xj, Yj — denoting that there is a bidirectional road connecting the cities numbered Xj and Yj.
Each of the following Q lines describes a query in one of the forms described earlier.

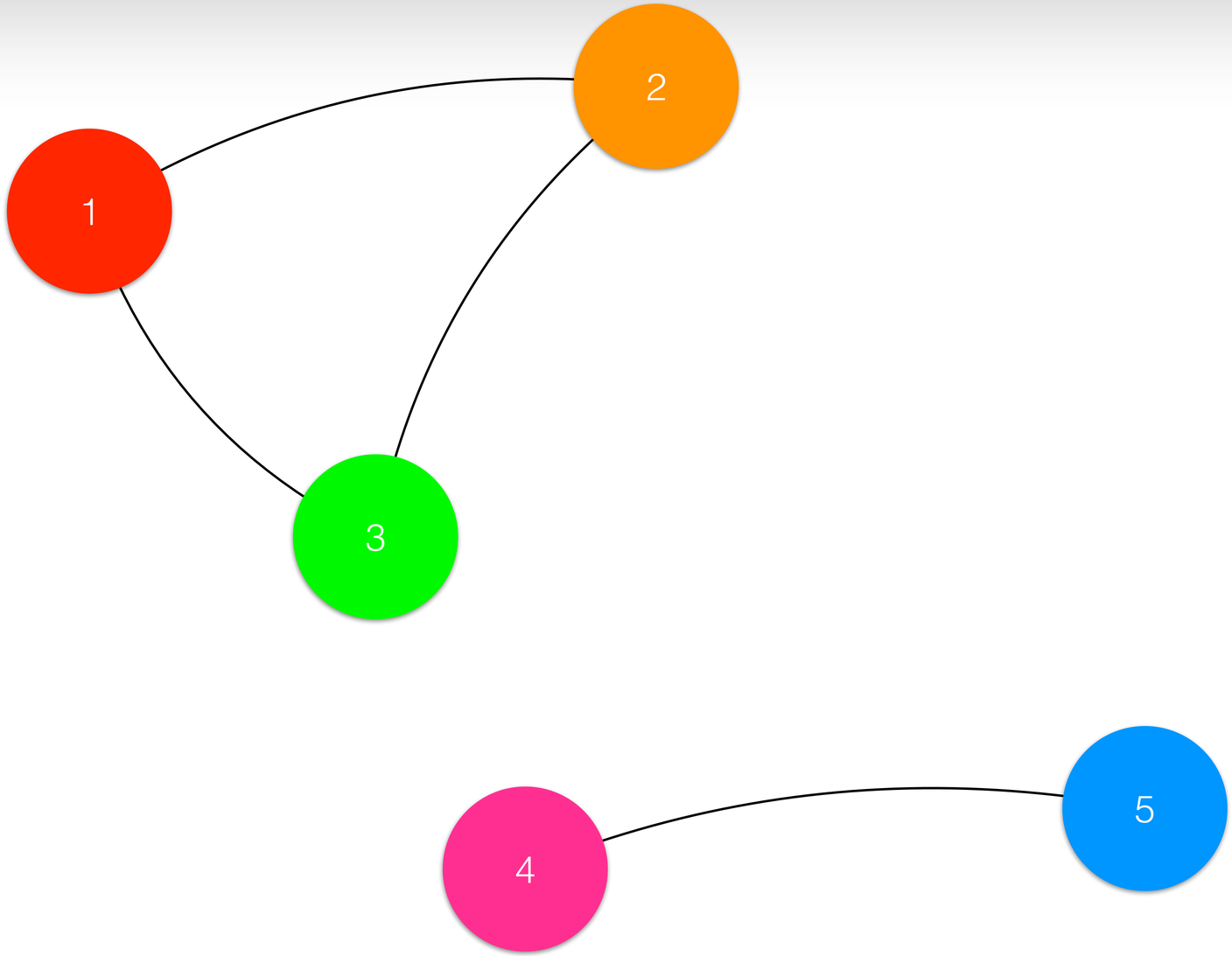
Output

Output Q lines. On the i^{th} line, output the size of the most populated region after performing i queries.

Constraints

- $1 \leq Xj, Yj \leq N$
- Roads' numbers are 1-indexed.
- There is no road that gets removed twice or more.
- $1 \leq Pi \leq 105$
- Subtask 1 (30 points) : $1 \leq N, M, Q \leq 103$
- Subtask 2 (70 points) : $1 \leq N, M, Q \leq 5 \times 105$

https://www.codechef.com/problems/ABROADS



https://www.codechef.com/problems/ABROADS