MUA

Maratones uniandes

GRAPH BASICS

ISIS 2801

Graphs

Graphs

- No recursive definition
- sets of data
- multidirectional relations * - *

$$G = \langle V, E \rangle$$

Graphs

Graphs

- No recursive definition
- sets of data
- multidirectional relations * - *

$$G = \langle V, E \rangle$$

Vertices set

Graphs

Graphs

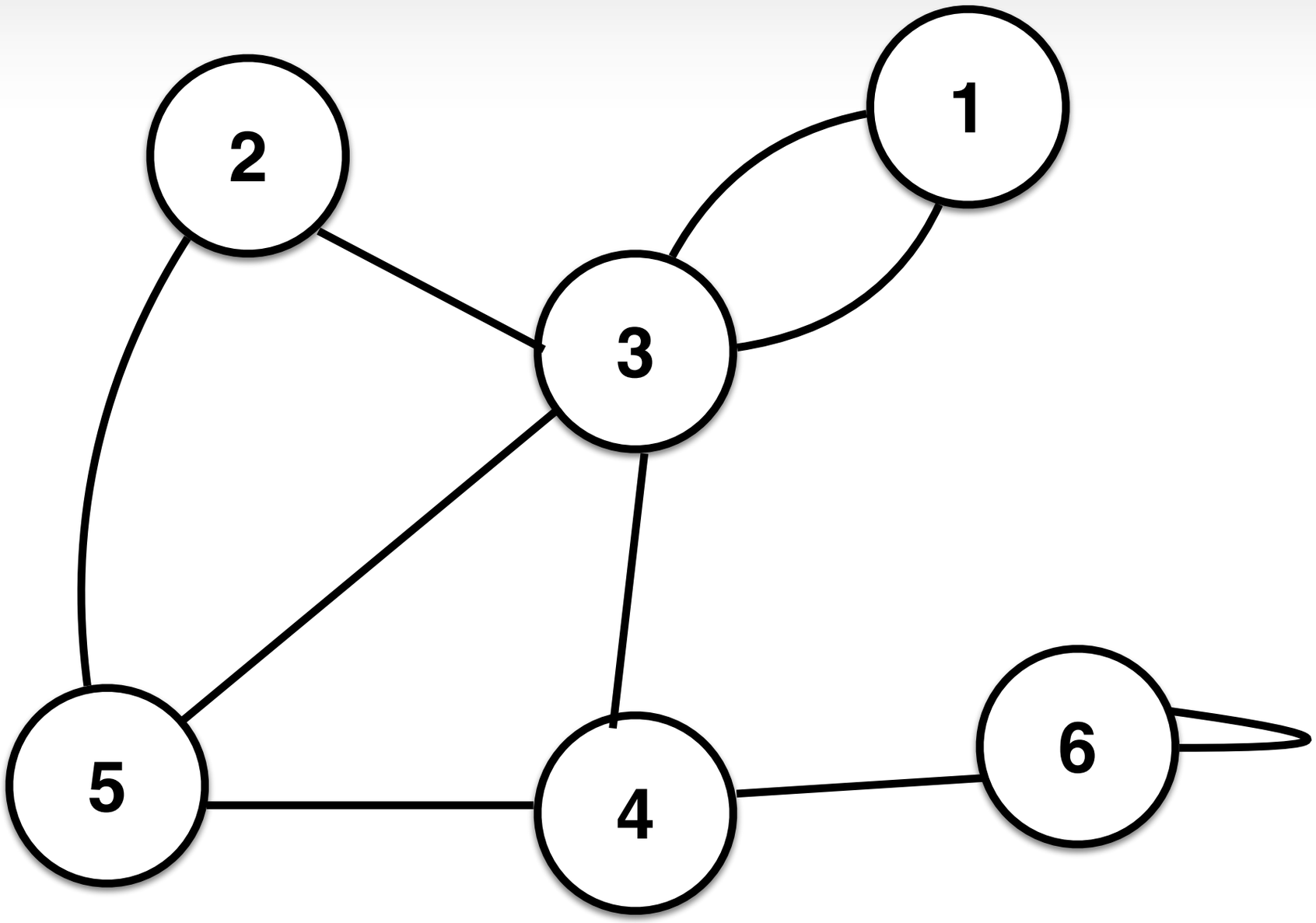
- No recursive definition
- sets of data
- multidirectional relations * - *

$$G = \langle V, E \rangle$$

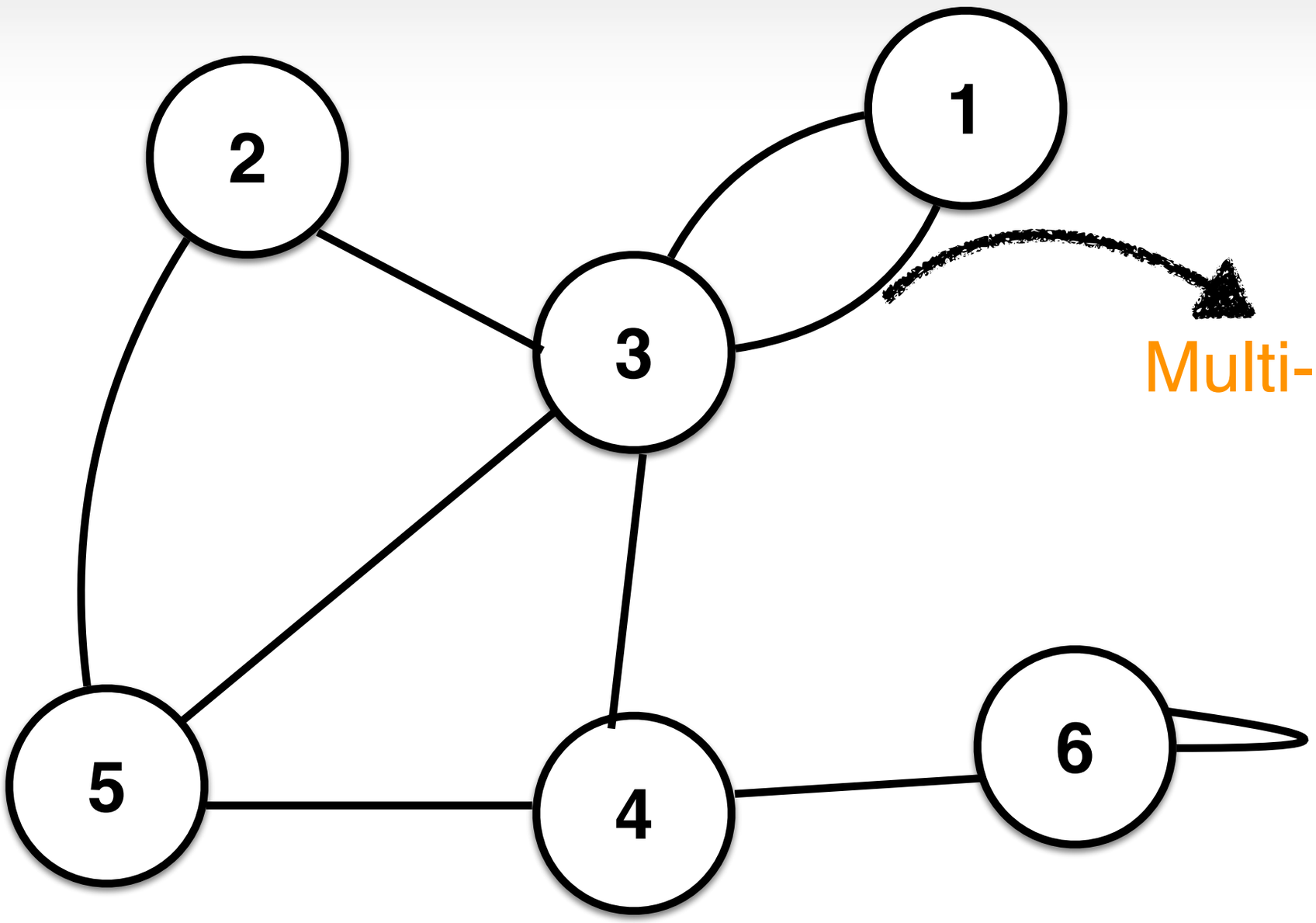
Vertices set

Edges set

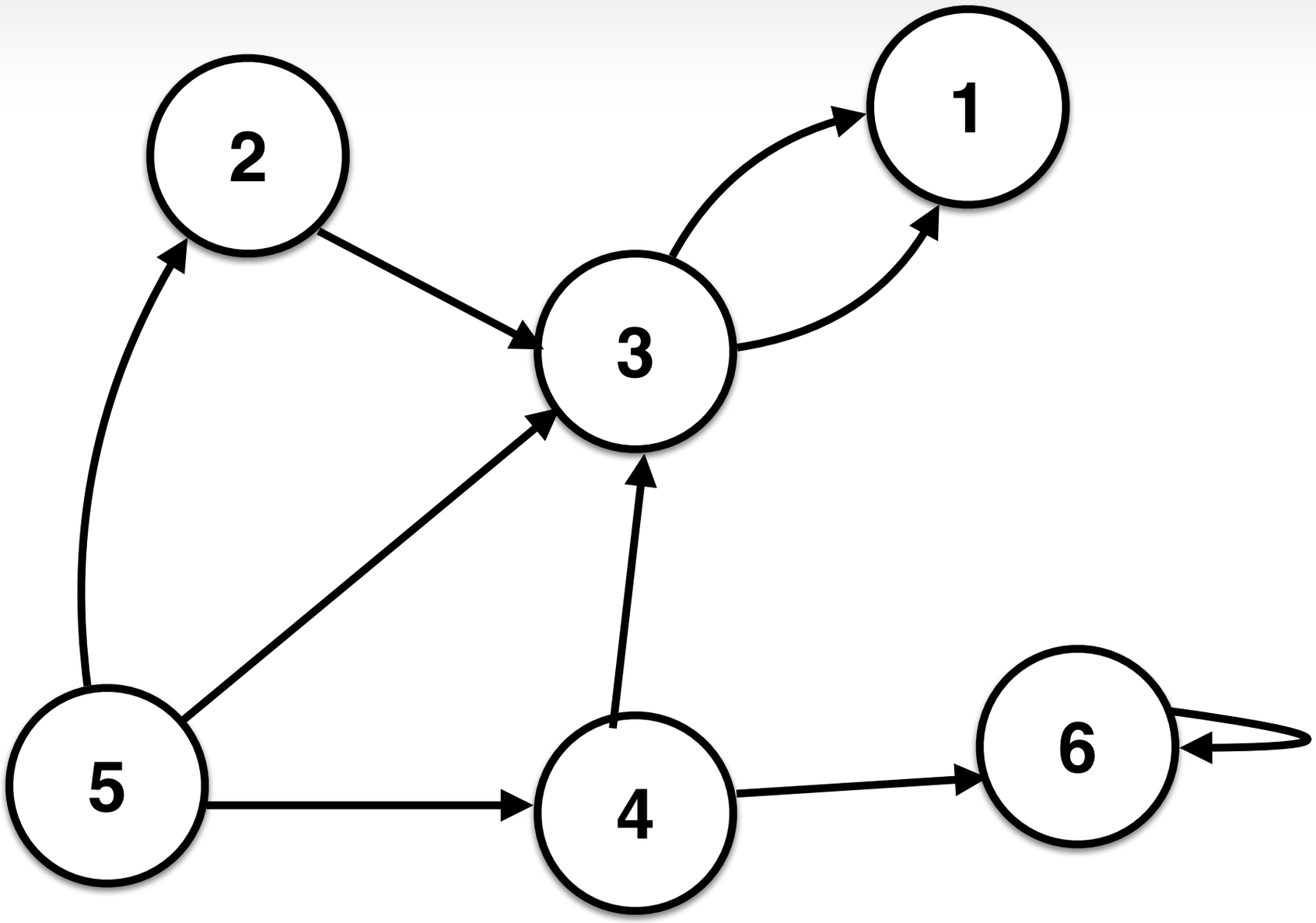
Graph



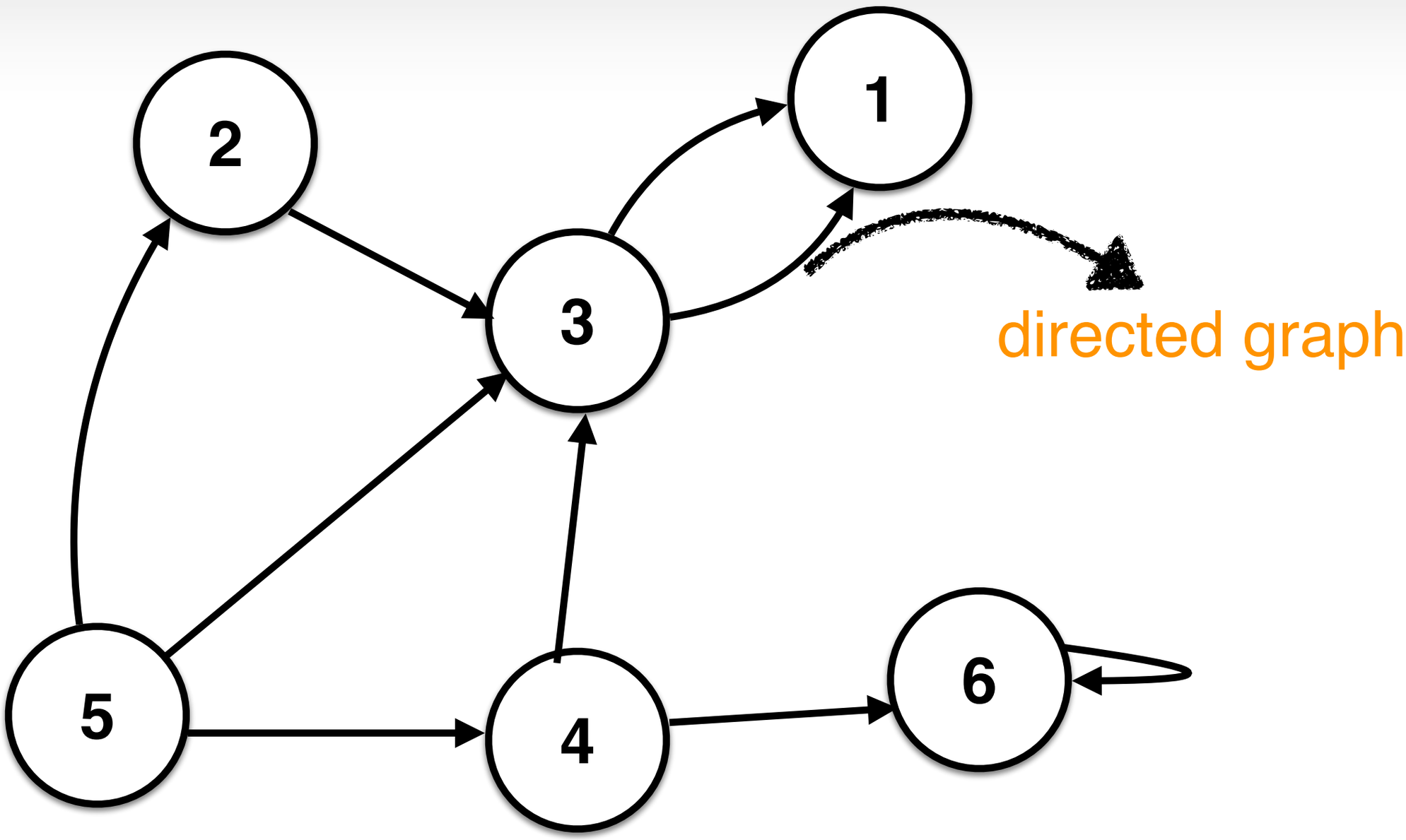
Graph



Graph



Graph



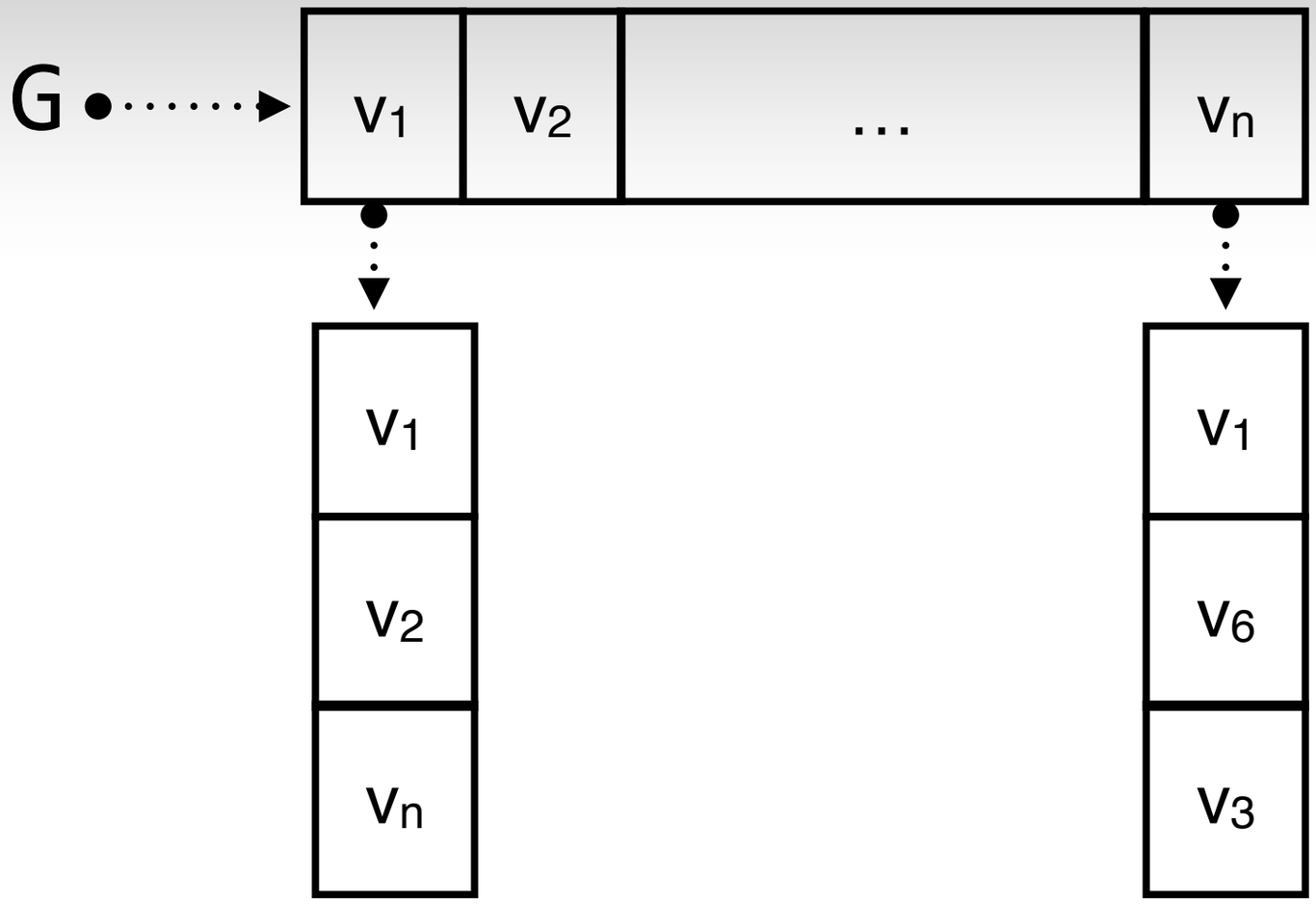
Adjacency matrix

G =

Vertices \ Vertices					
	-	-	-	-	-
	-	1	-	1	-
	-	-	-	-	-
	-	1	-	-	1
	-	-	-	1	-

- Too much memory
- Repeats information

Adjacency lists



- Information replication (in multigraphs)

Adjacency lists

```
#include <bits/stdc++.h>

using namespace std;

void addEdge(vector<int> nodes[], int e1, int e2) {
    nodes[e1].push_back(e2);
    nodes[e2].push_back(e1);
}

int main() {
    int n, nEdges, e1, e2;
    cin >> n;
    vector<int> nodes[n];
    cin >> nEdges;
    for(int i=0; i<nEdges; i++) {
        cin >> e1 >> e2;
        addEdge(nodes, e1, e2);
    }
    printGraph(nodes);
    return 0;
}
```

Sparse adjacency matrix

G =

-	-	-	-	-
-	1	-	1	-
-	-	-	-	-
-	1	-	-	1
-	-	-	1	-

```
graph = {(1,1):1, (1,3):1, (3,1):1, (3,4):1}
graph[(4,3)] = 1
graph[(0,0)]                #Error
graph.get((0,0))            #Nothing
```

Sparse adjacency matrix

```
def main():
    nodes = {}
    n = int(stdin.readline())
    edgeNum = int(stdin.readline())
    for i in range(0, edgeNum):
        edges = stdin.readline().split()
        nodes[int(edges[0]), int(edges[1])] = 1
    printGraph(nodes, n)
```

Sparse adjacency matrix

```
def adjacent_edges(N):
  for i in range(N):
    for j in range(N):
      if (i,j) in graph: print(‘{} adjacent to
      {}’.format(i,j))
```

Graphs

Paths

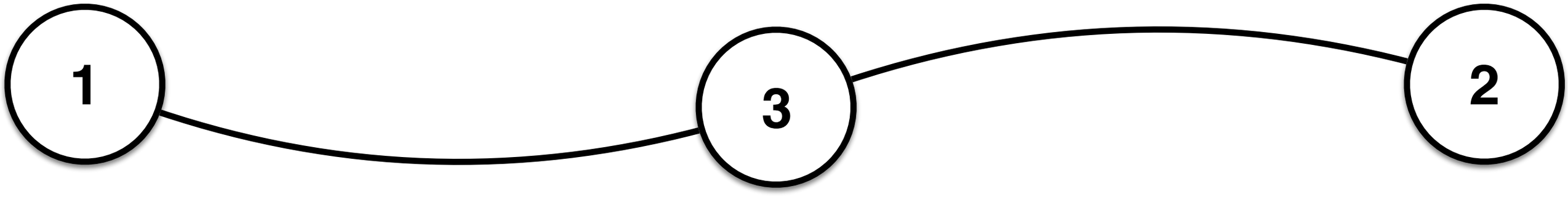
node ω is reachable **in one step** from node υ

if ∃ e ∈ E such that (υ , e, ω)

node ω is reachable from node υ if ∃ e_1, … $e_n \in E$

and υ_1, … $\upsilon_n \in V$

such that ($\upsilon = \upsilon_1$, e_1, υ_2), … , (υ_{n-1} , e_1, υ_n $\upsilon = \omega$)



Paths

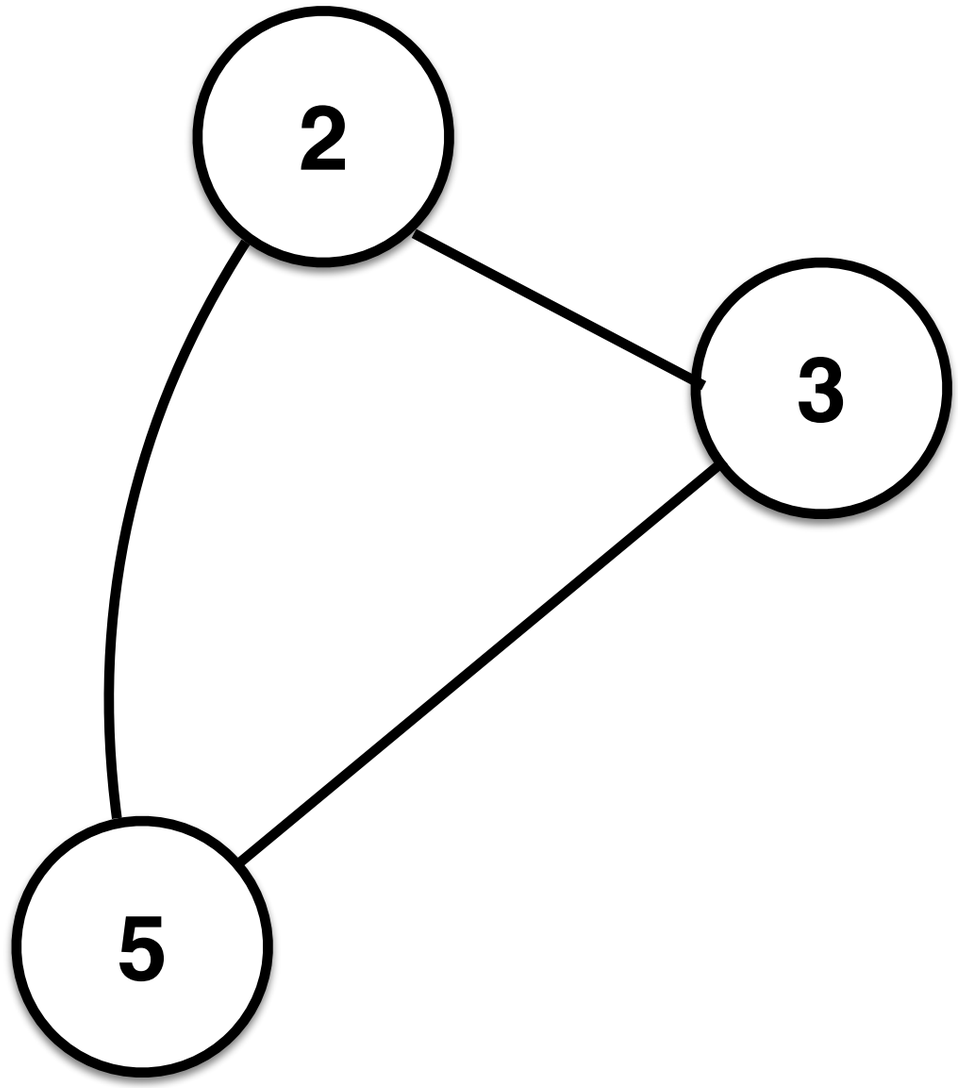
How to find if there is a path between nodes ω and υ

Graphs

Connectivity A graph is **connected**, if it is possible to find a path **between any two pair of nodes**

Graphs

Cycles A cycle is a path in which a node is visited more than once



https://codeforces.com/problemset/problem/1055/A

Alice has a birthday today, so she invited home her best friend Bob. Now Bob needs to find a way to commute to the Alice's home.

In the city in which Alice and Bob live, the first metro line is being built. This metro line contains n stations numbered from 1 to n. Bob lives near the station with number 1, while Alice lives near the station with number s. The metro line has two tracks. Trains on the first track go from the station 1 to the station n and trains on the second track go in reverse direction. Just after the train arrives to the end of its track, it goes to the depot immediately, so it is impossible to travel on it after that.

Some stations are not yet open at all and some are only partially open — for each station and for each track it is known whether the station is closed for that track or not. If a station is closed for some track, all trains going in this track's direction pass the station without stopping on it.

When Bob got the information on opened and closed stations, he found that traveling by metro may be unexpectedly complicated. Help Bob determine whether he can travel to the Alice's home by metro or he should search for some other transport.

https://codeforces.com/problemset/problem/1055/A

Input

The **first line contains two integers** n **and** s **(2≤**s**≤**n**≤1000)** — the number of stations in the metro and the number of the station where Alice's home is located. Bob lives at station 1.

Next lines describe information about closed and open stations.

The **second line contains** n **integers** $a1, a2, \ldots, an$ **(**a_i**=0 or** a_i**=1)**. If a_i**=1, then the** i**-th station is open** on the first track (that is, in the direction of increasing station numbers). Otherwise the station is closed on the first track.

The **third line contains** n **integers** $b1, b2, \ldots, bn$ **(**bi**=0 or** bi**=1)**. If bi=1, then the i-th station is open on the second track (that is, in the direction of decreasing station numbers). Otherwise the station is closed on the second track.

https://codeforces.com/problemset/problem/1055/A

Output

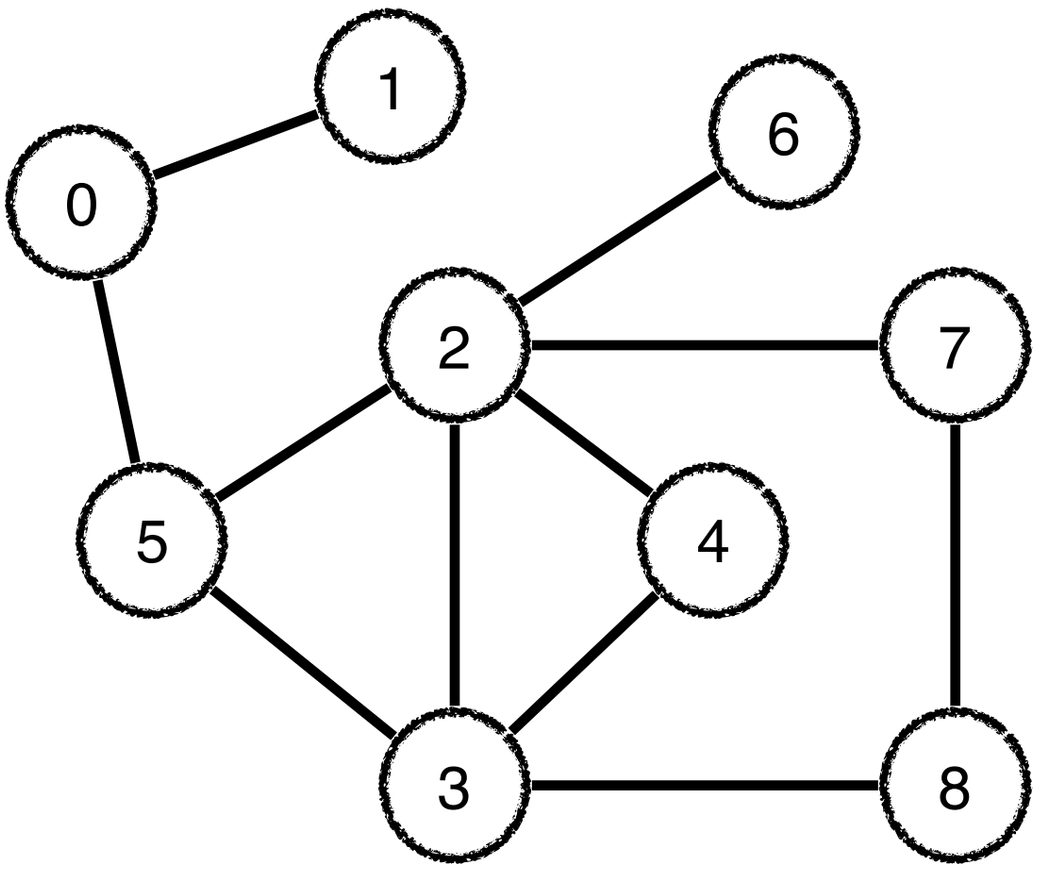
Print "**YES**" (quotes for clarity) if Bob will be able to commute to the Alice's home by metro and "**NO**" (quotes for clarity) otherwise.

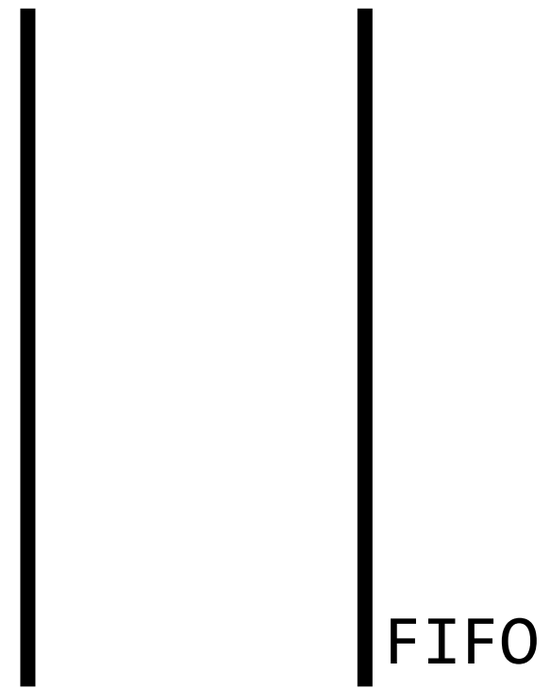
You can print each letter in any case (upper or lower).

https://codeforces.com/problemset/problem/1055/A

PATHS

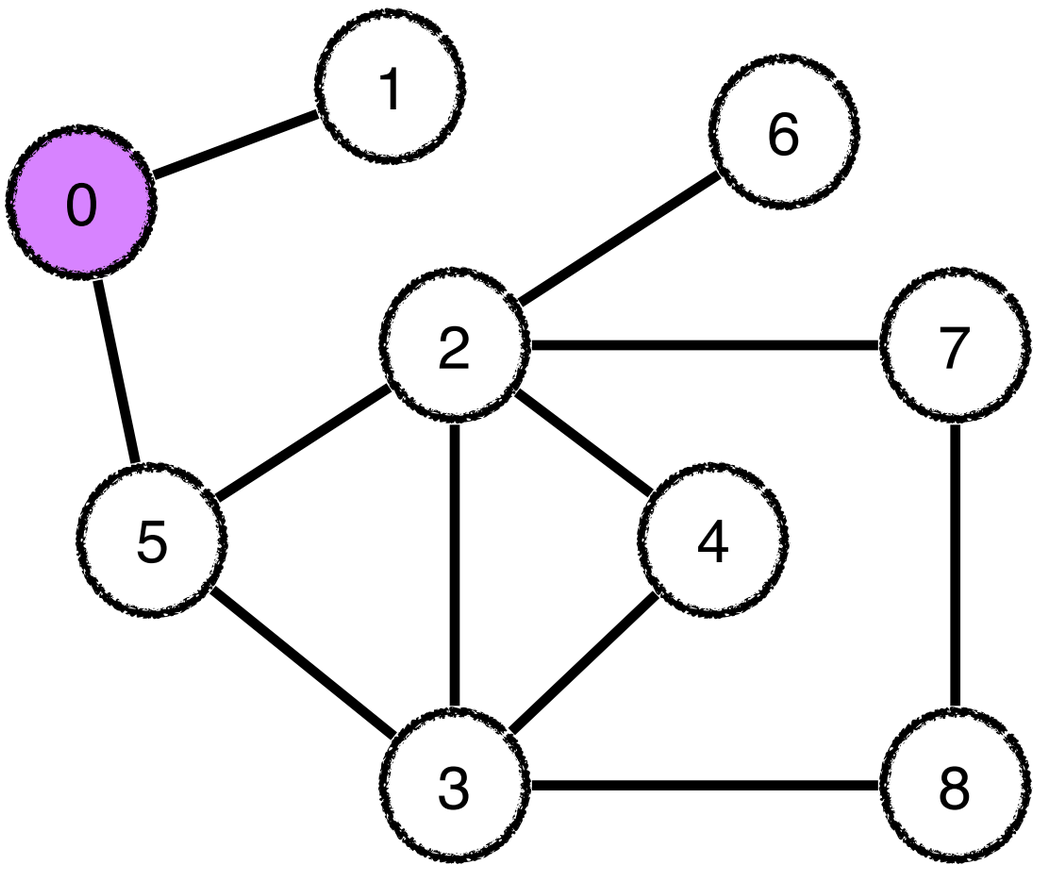
Breadth-first search BFS





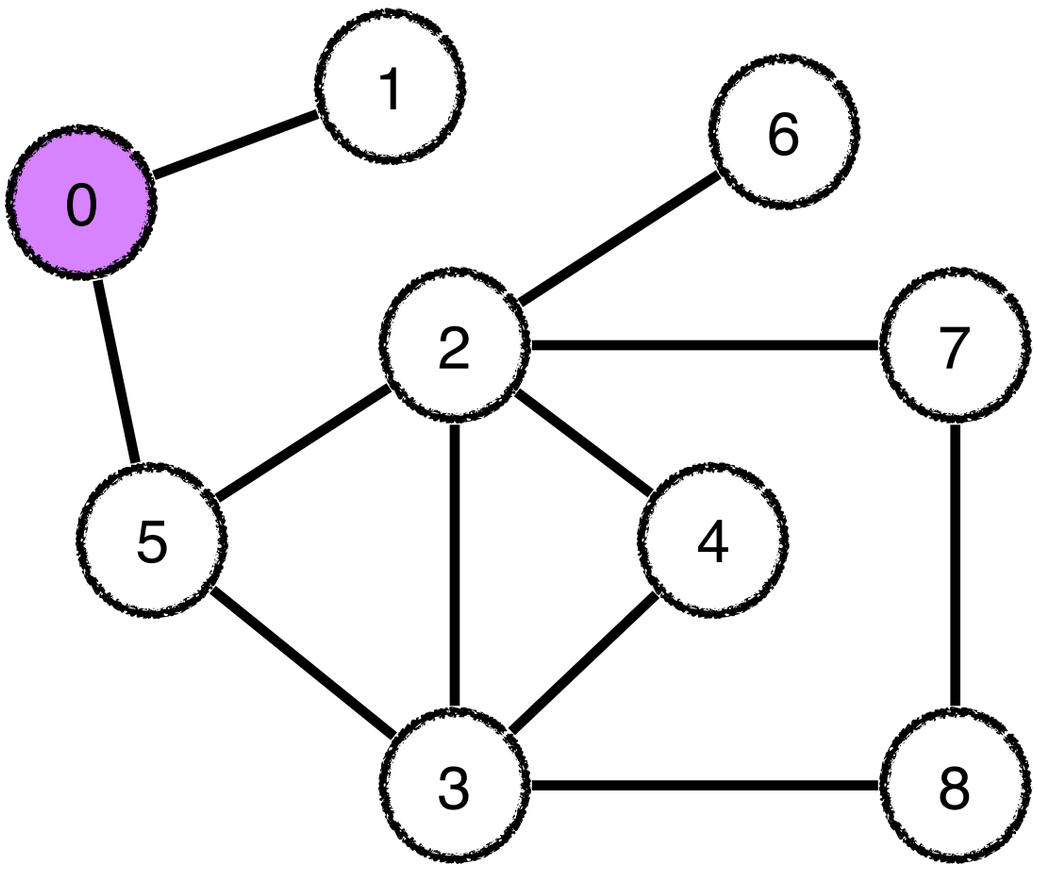
Added:

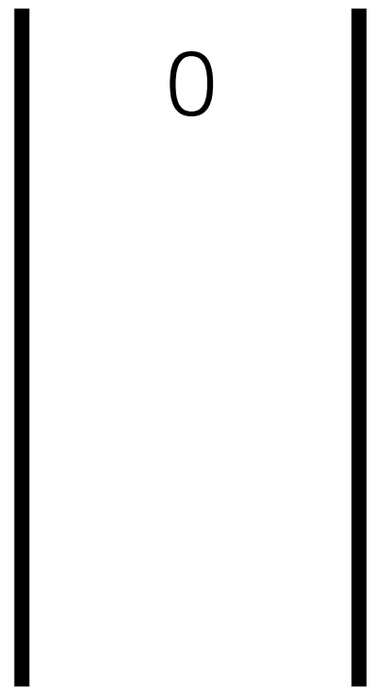
BFS



Added:

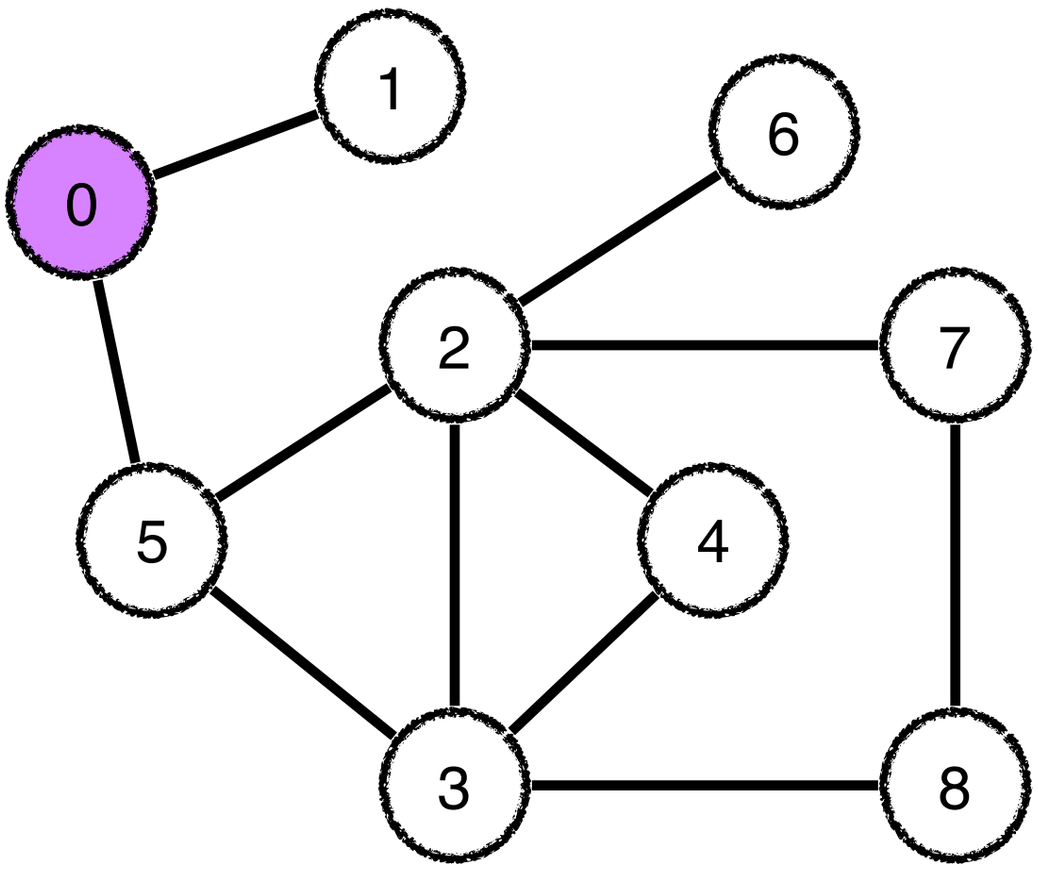
BFS

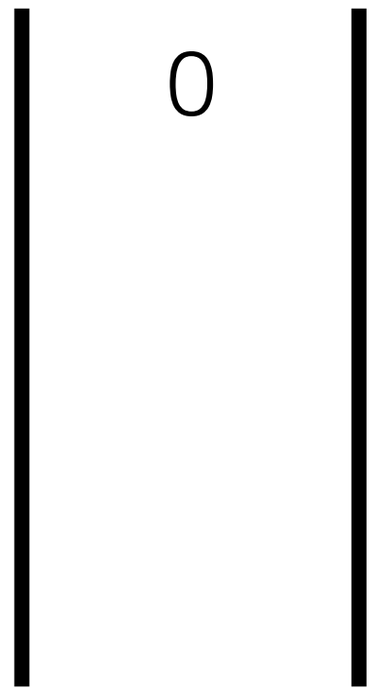




Added:

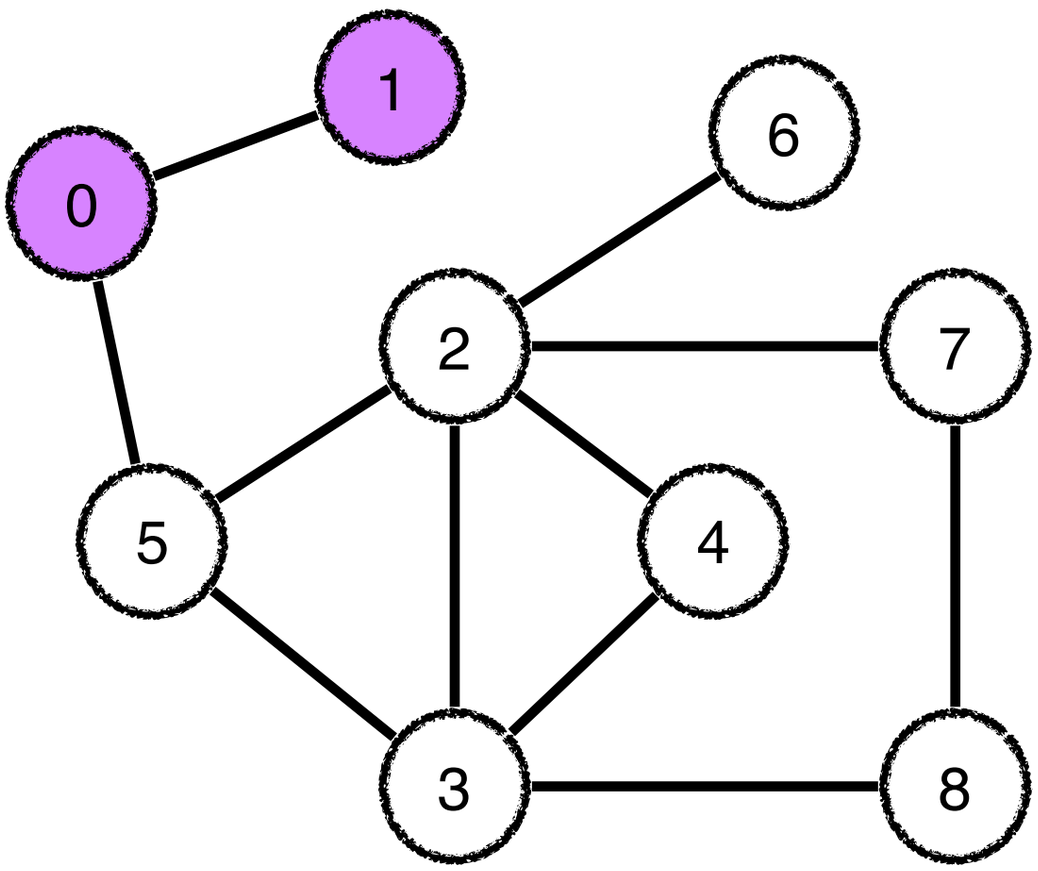
BFS

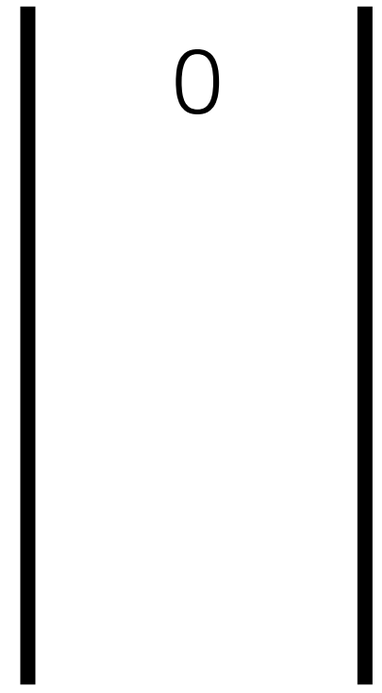




Added: 0

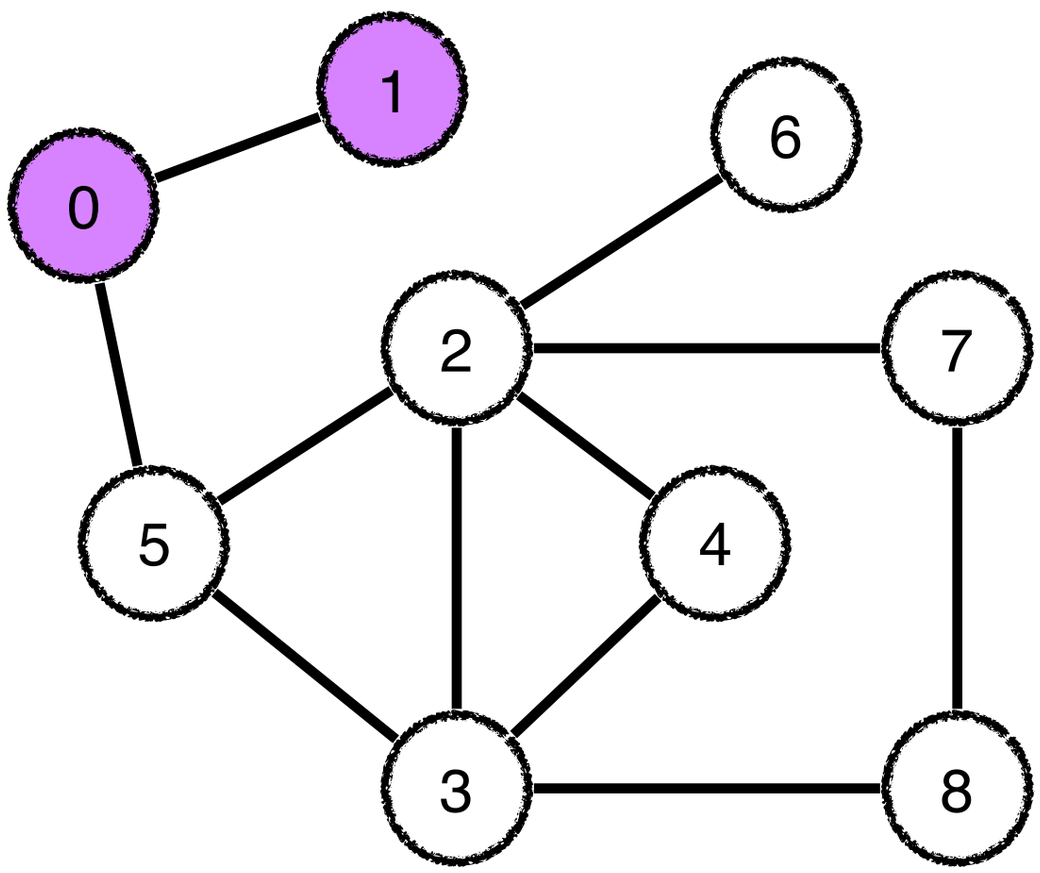
BFS

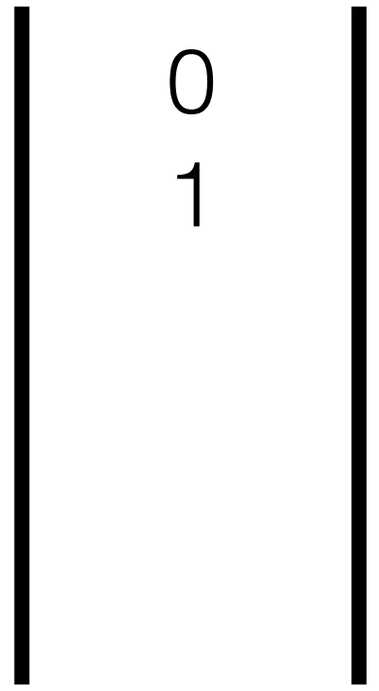




Added: 0

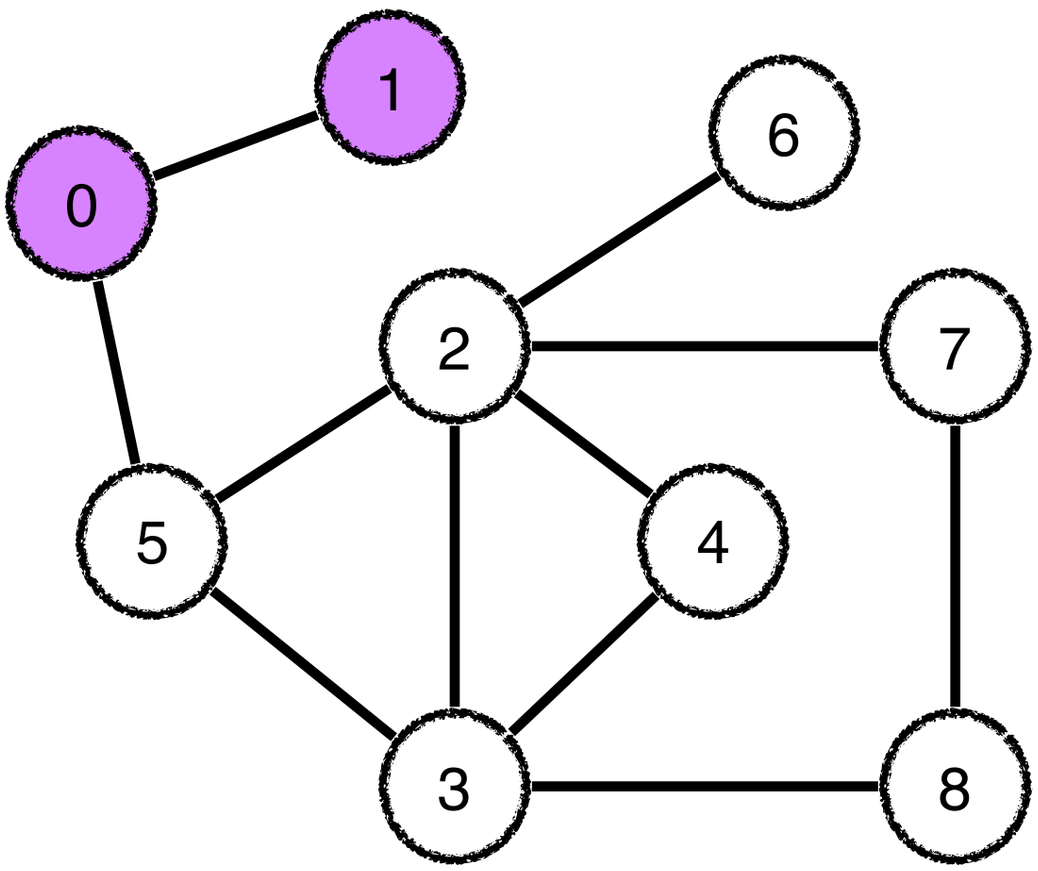
BFS

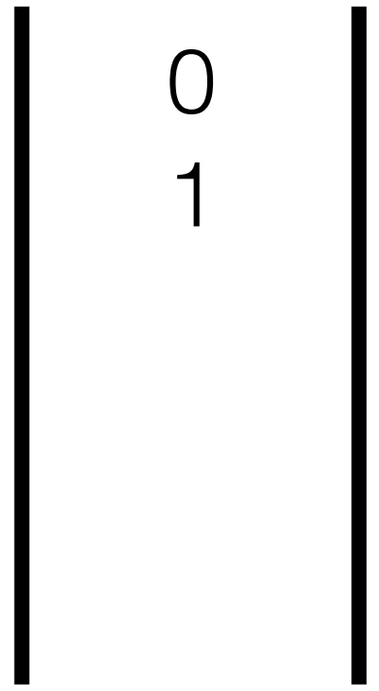




Added: 0

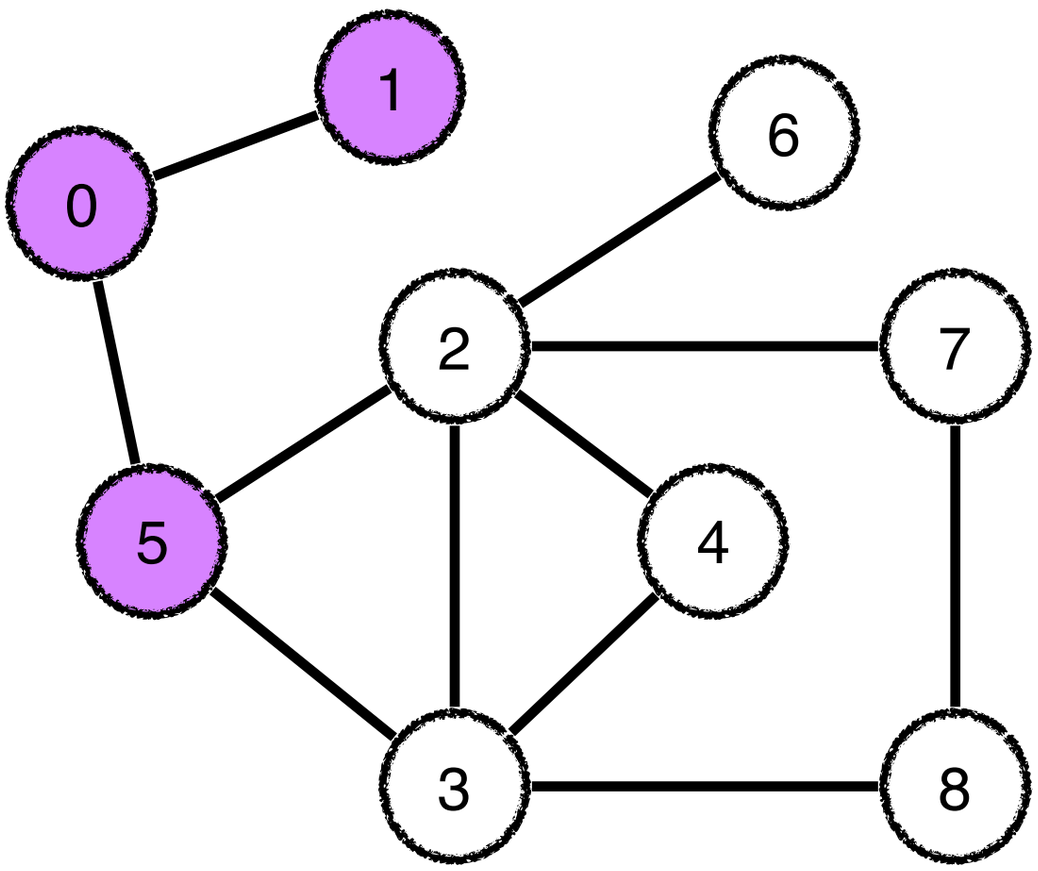
BFS

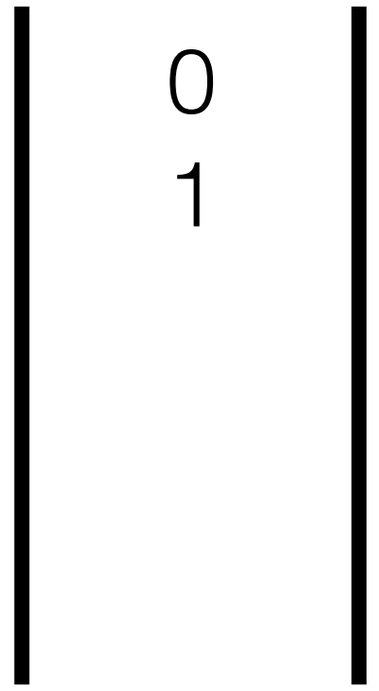




Added: 0 1

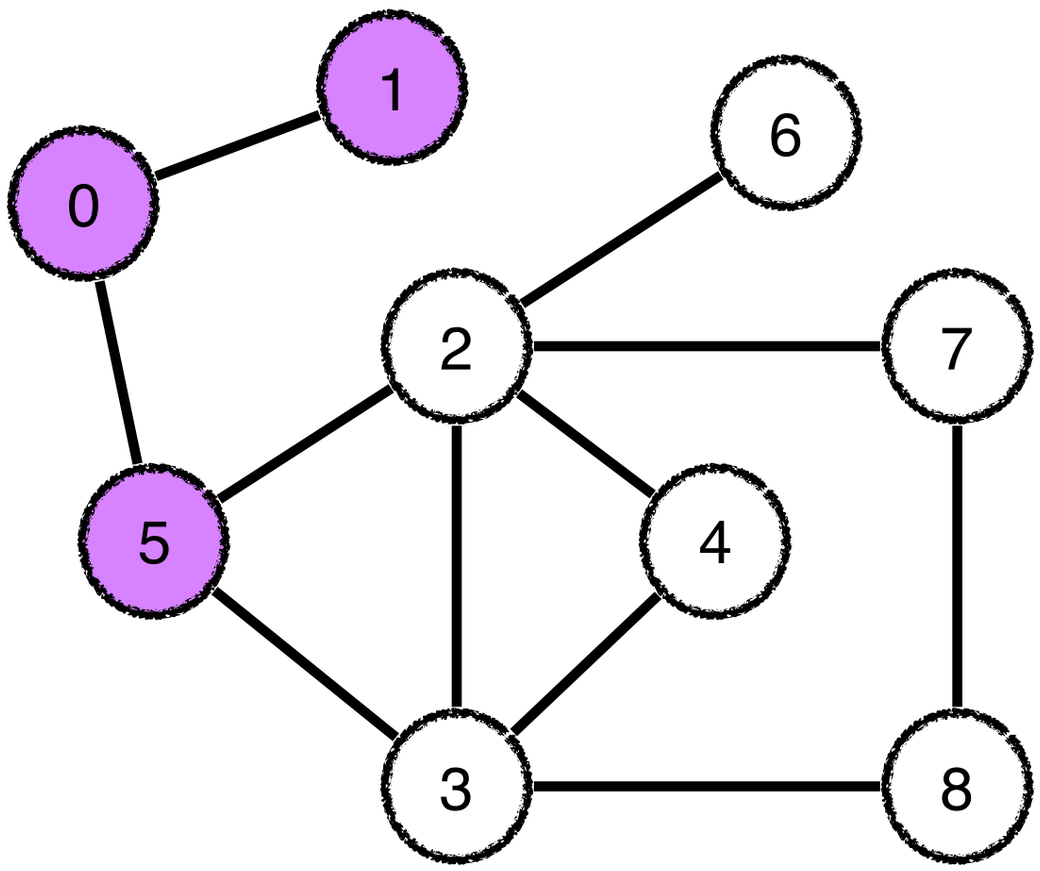
BFS

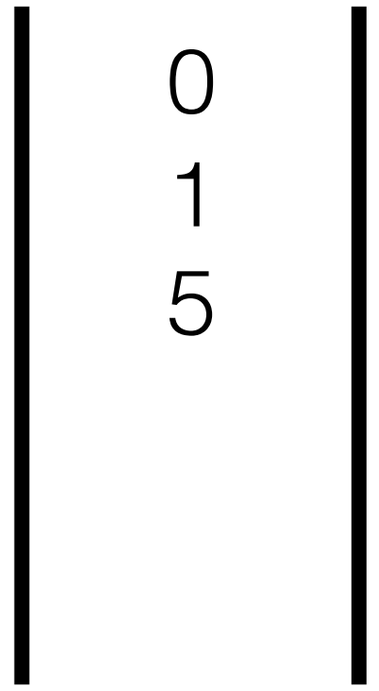




Added: 0 1

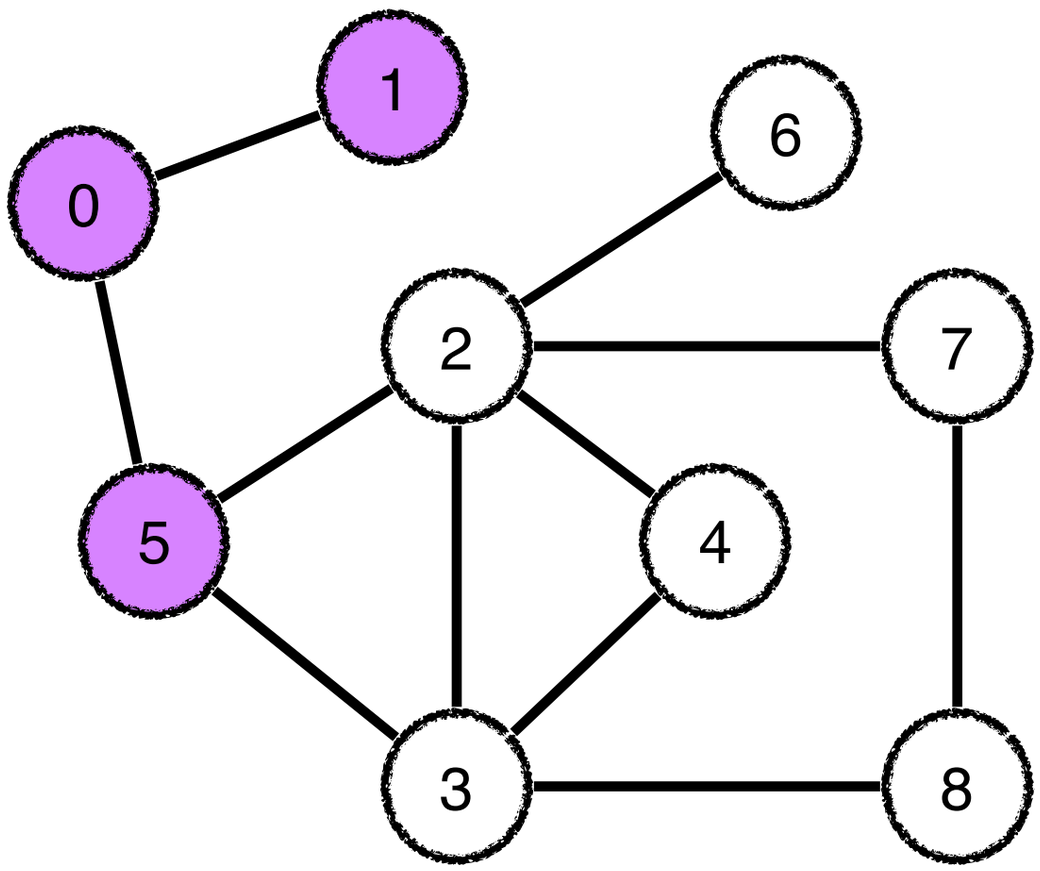
BFS

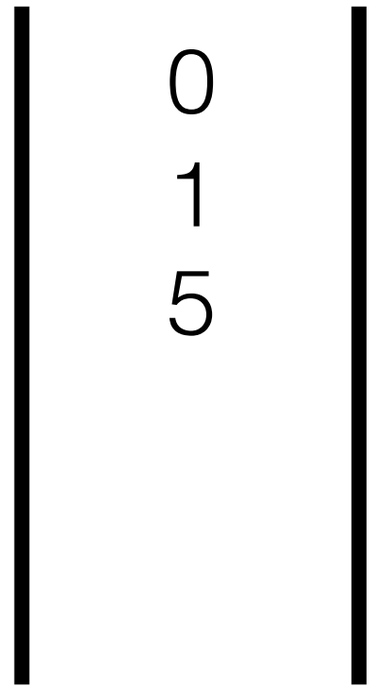




Added: 0 1

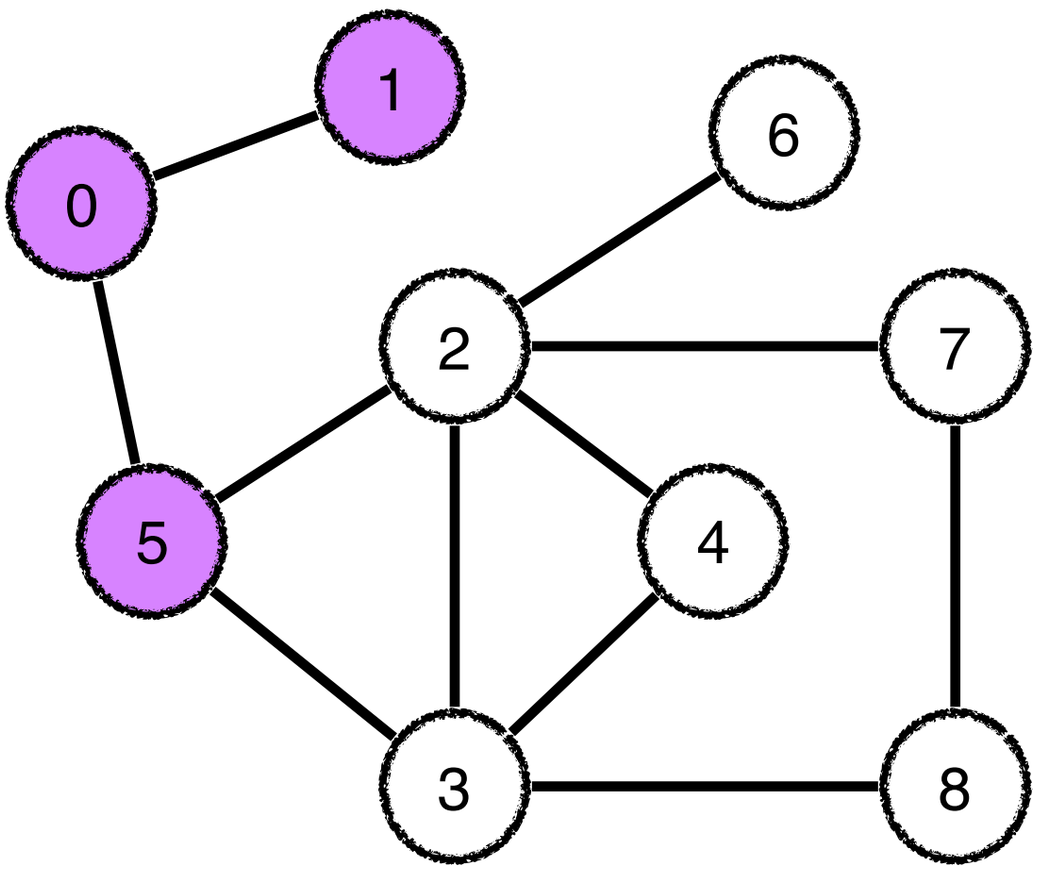
BFS

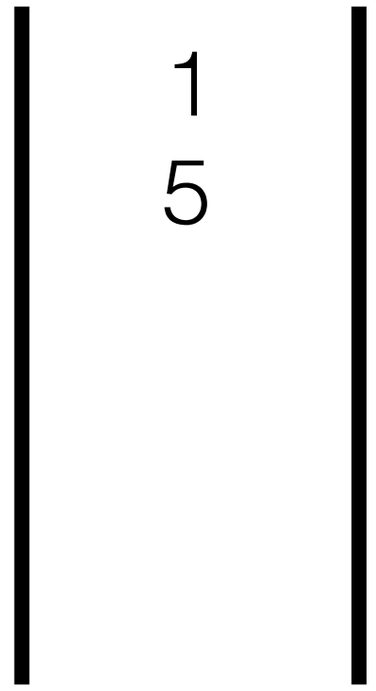




Added: 0 1 5

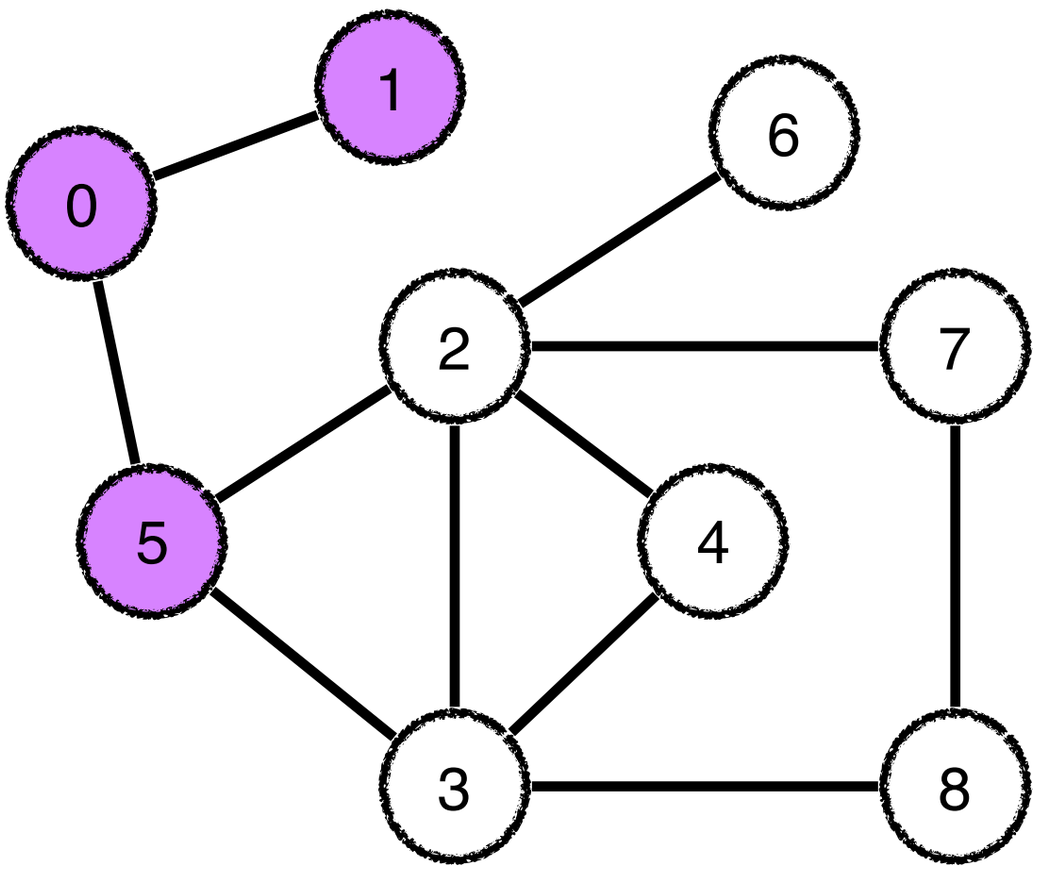
BFS

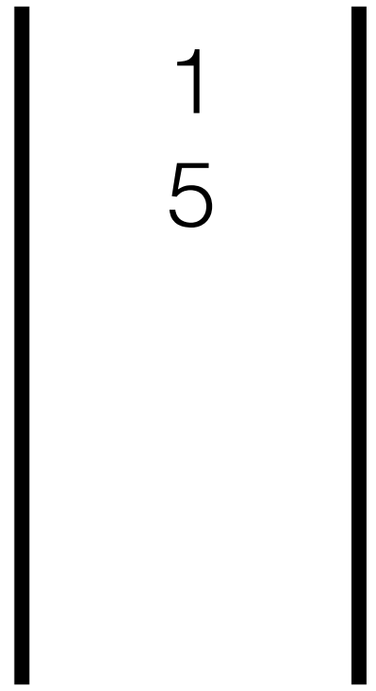




Added: 0 1 5

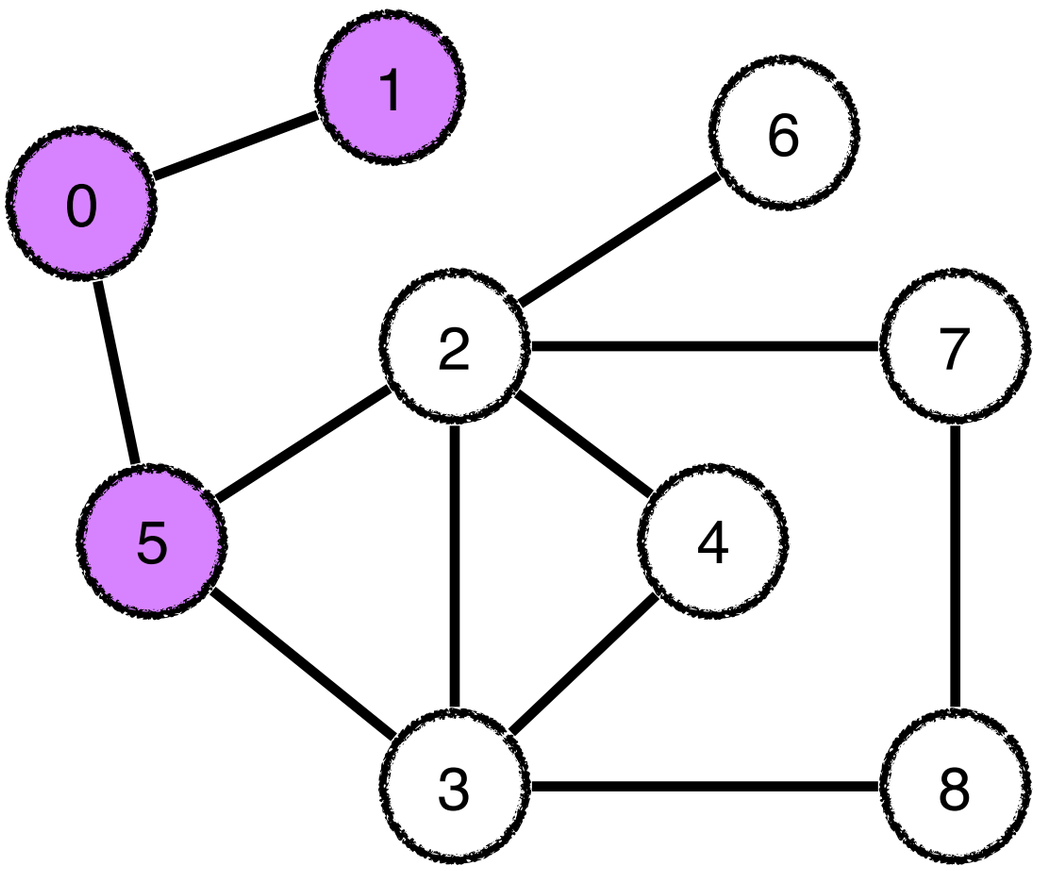
BFS

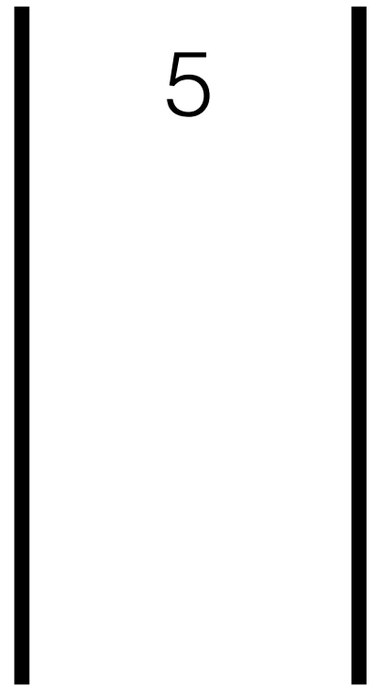




Added: 0 1 5

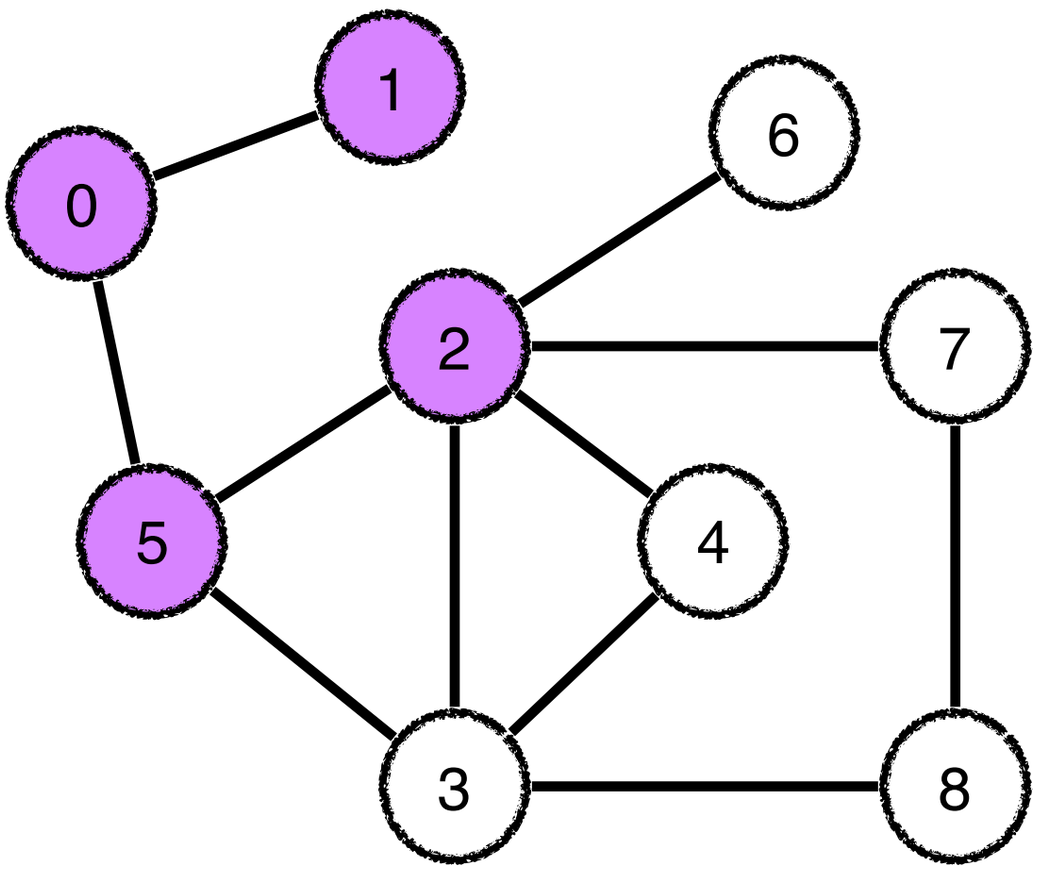
BFS

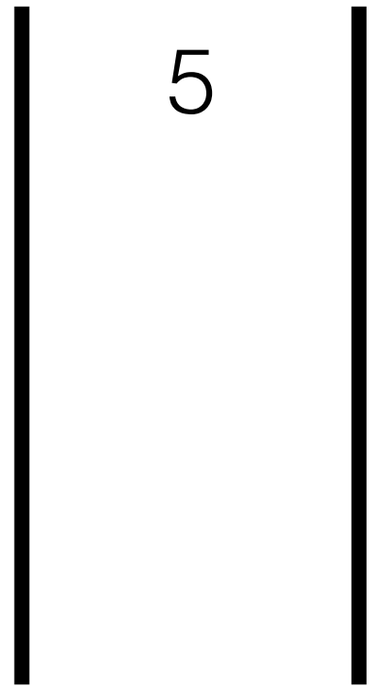




Added: 0 1 5

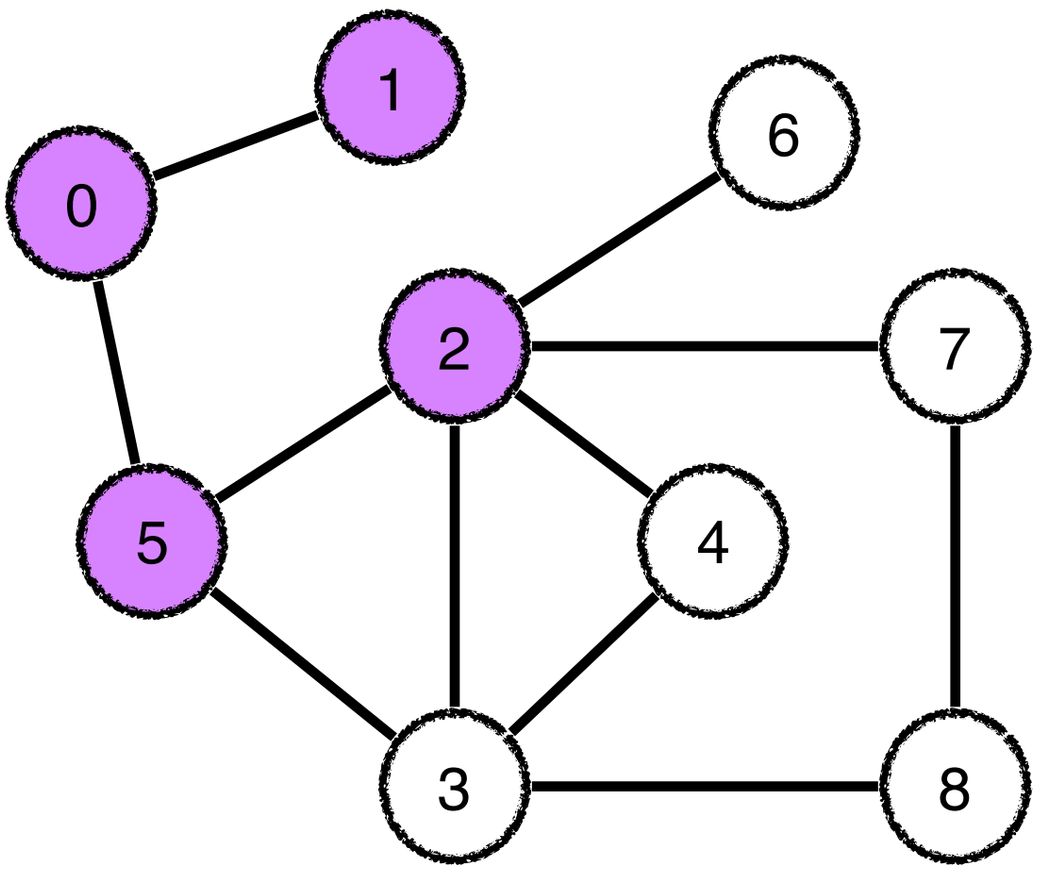
BFS

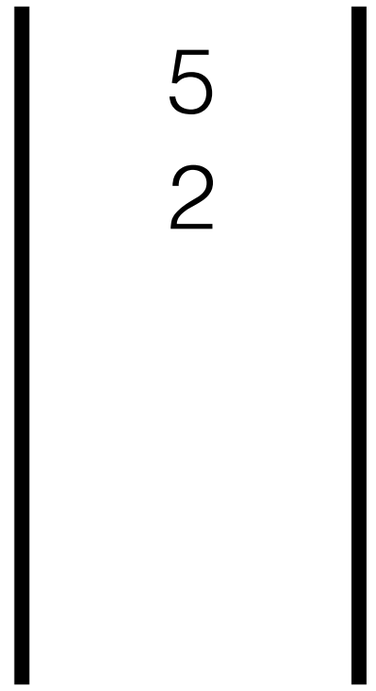




Added: 0 1 5

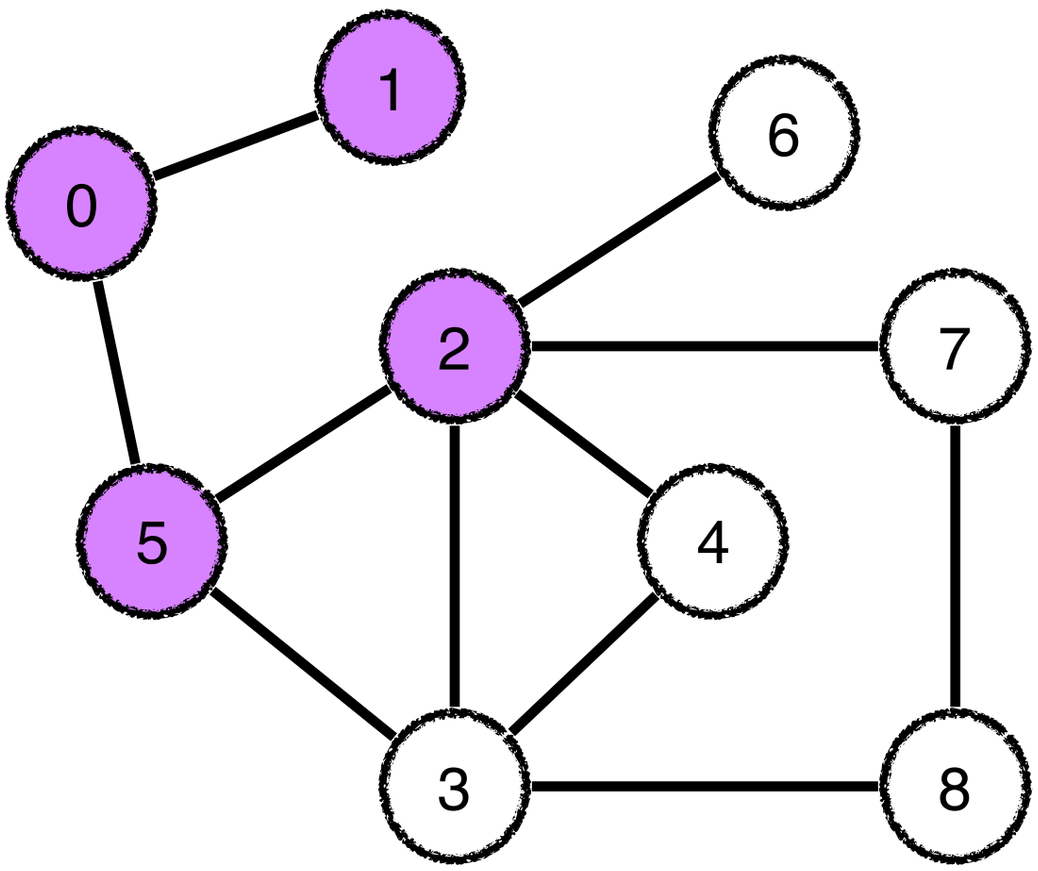
BFS

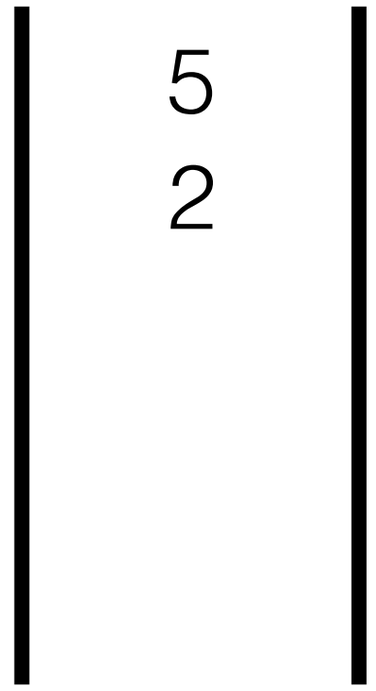




Added: 0 1 5

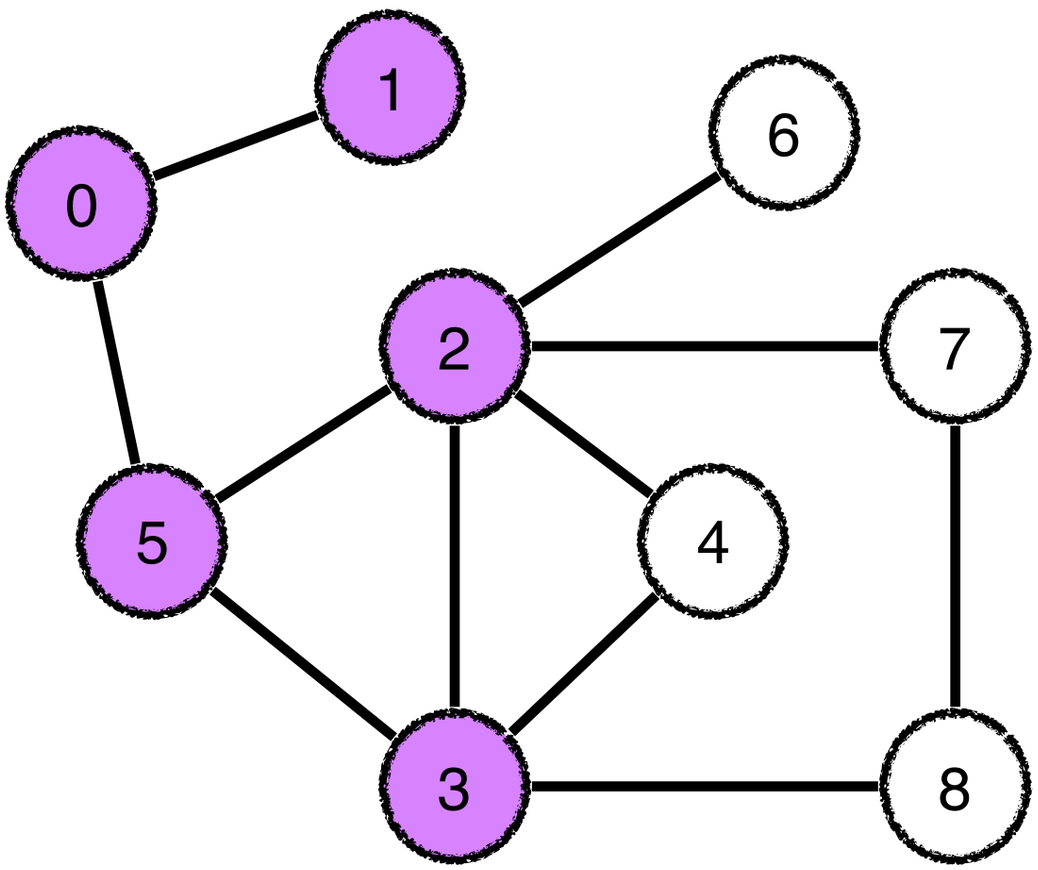
BFS

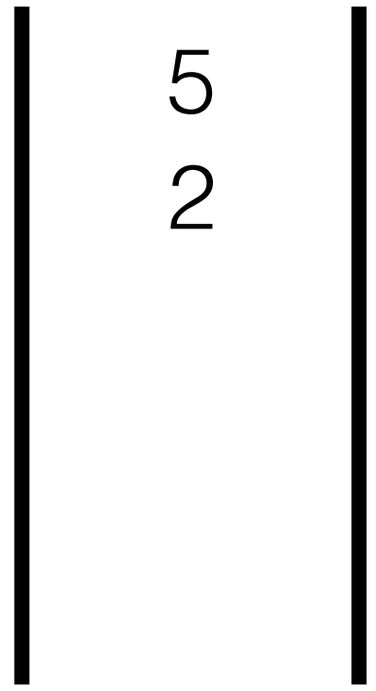




Added: 0 1 5 2

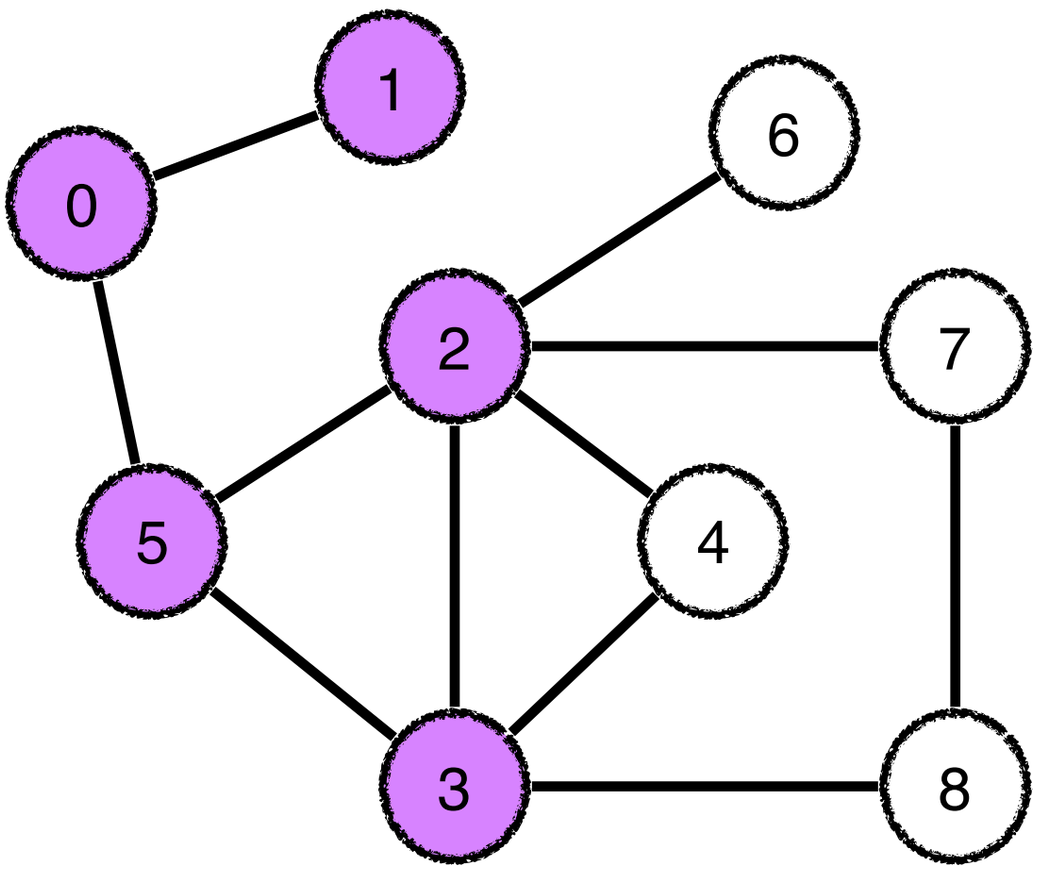
BFS

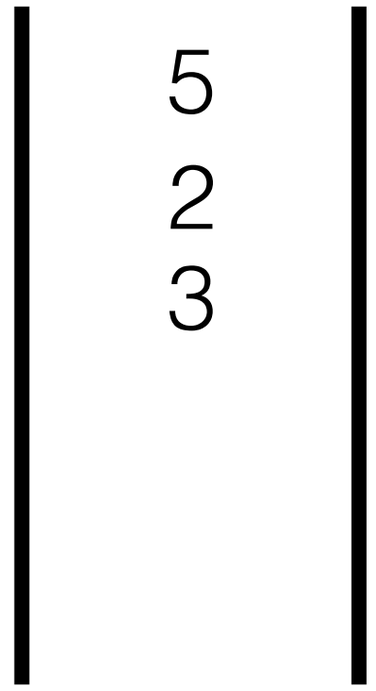




Added: 0 1 5 2

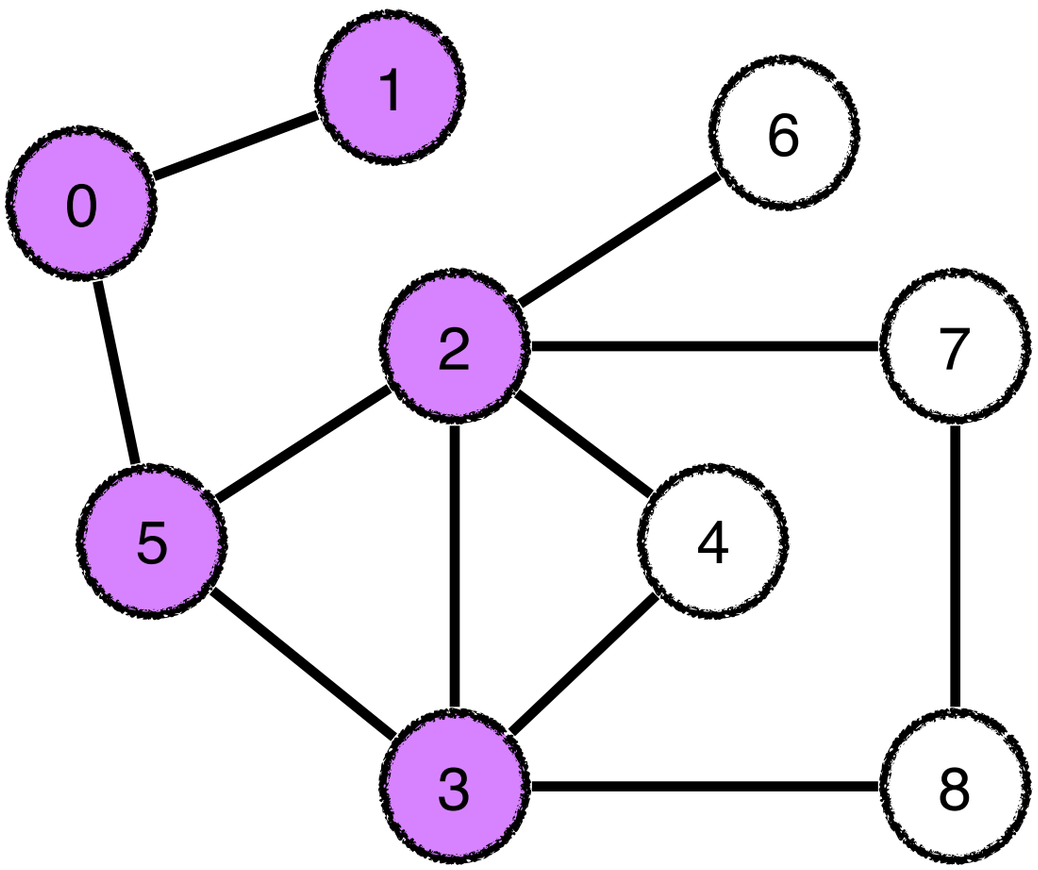
BFS

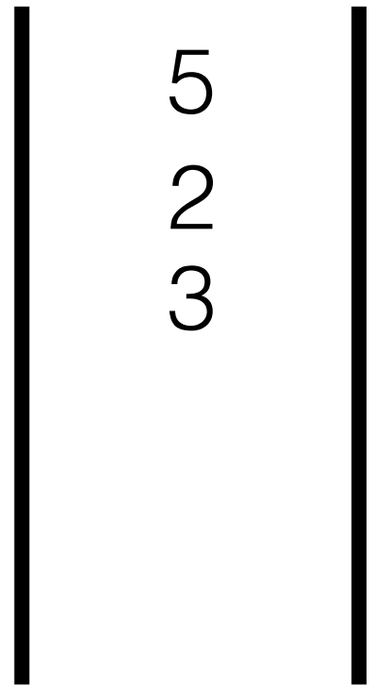




Added: 0 1 5 2

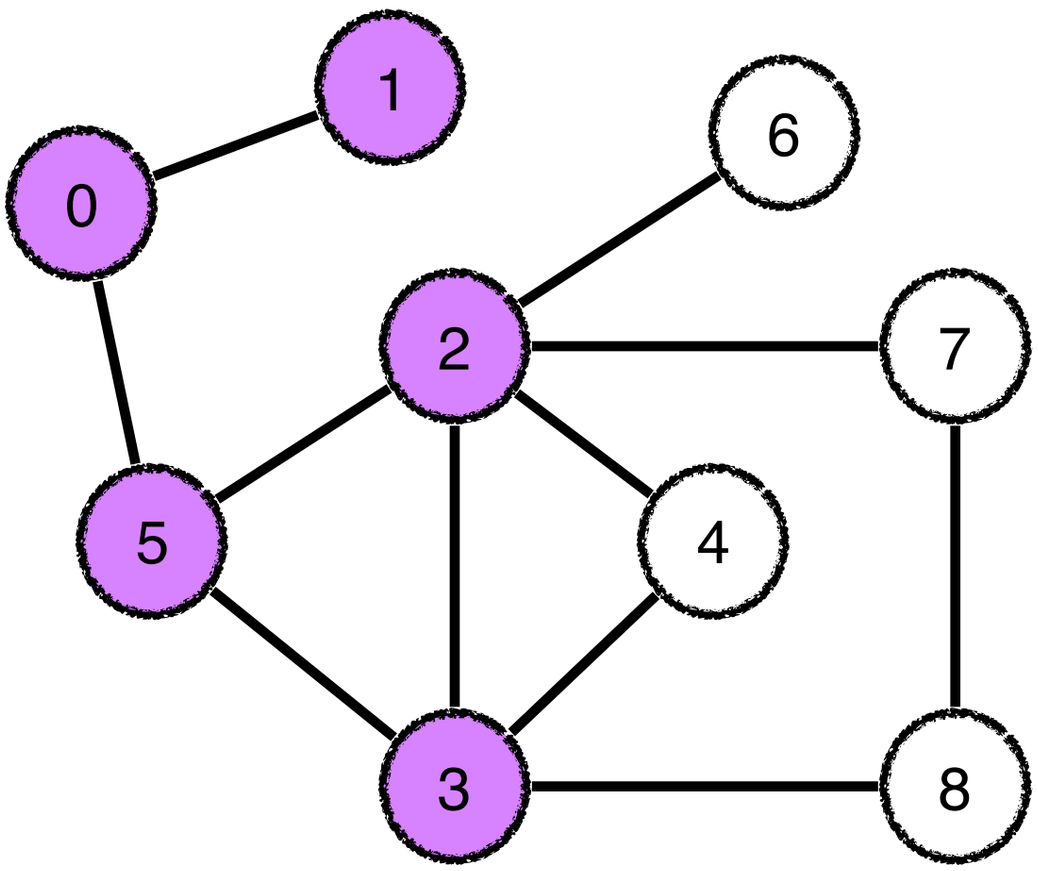
BFS

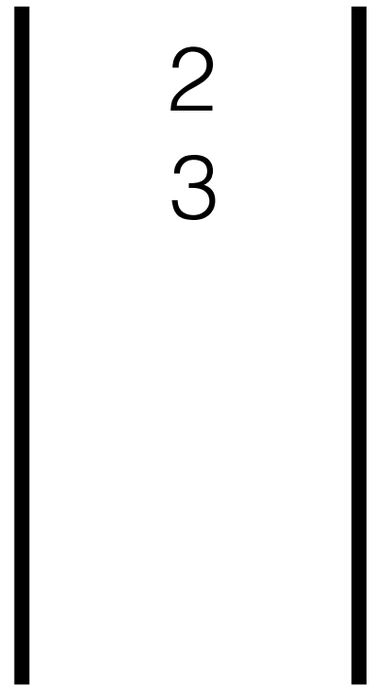




Added: 0 1 5 2 3

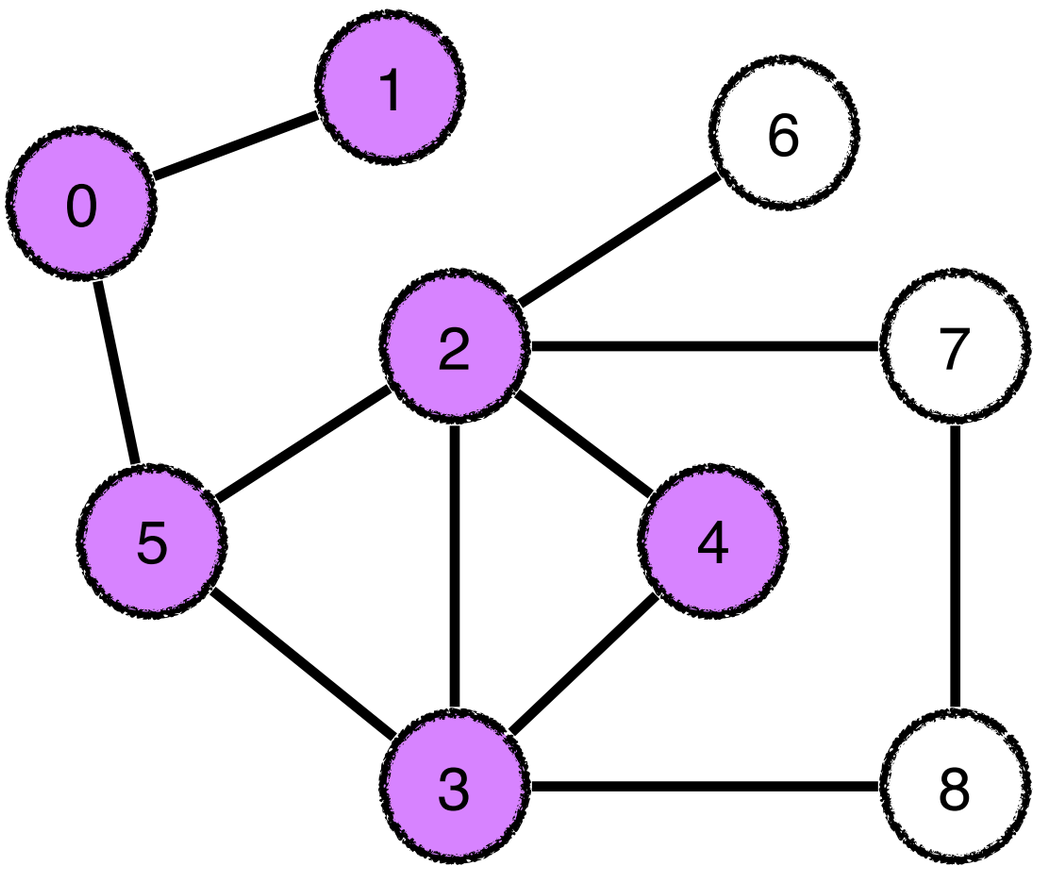
BFS

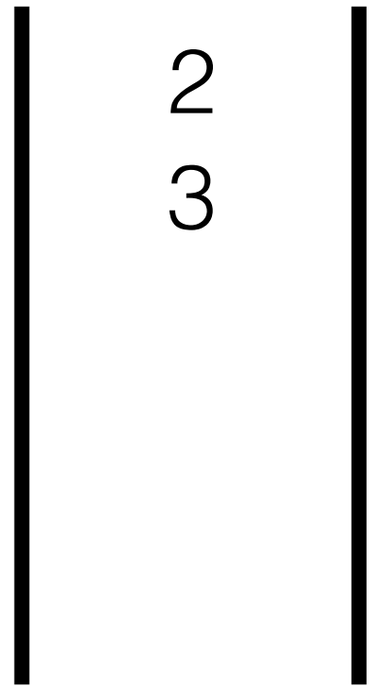




Added: 0 1 5 2 3

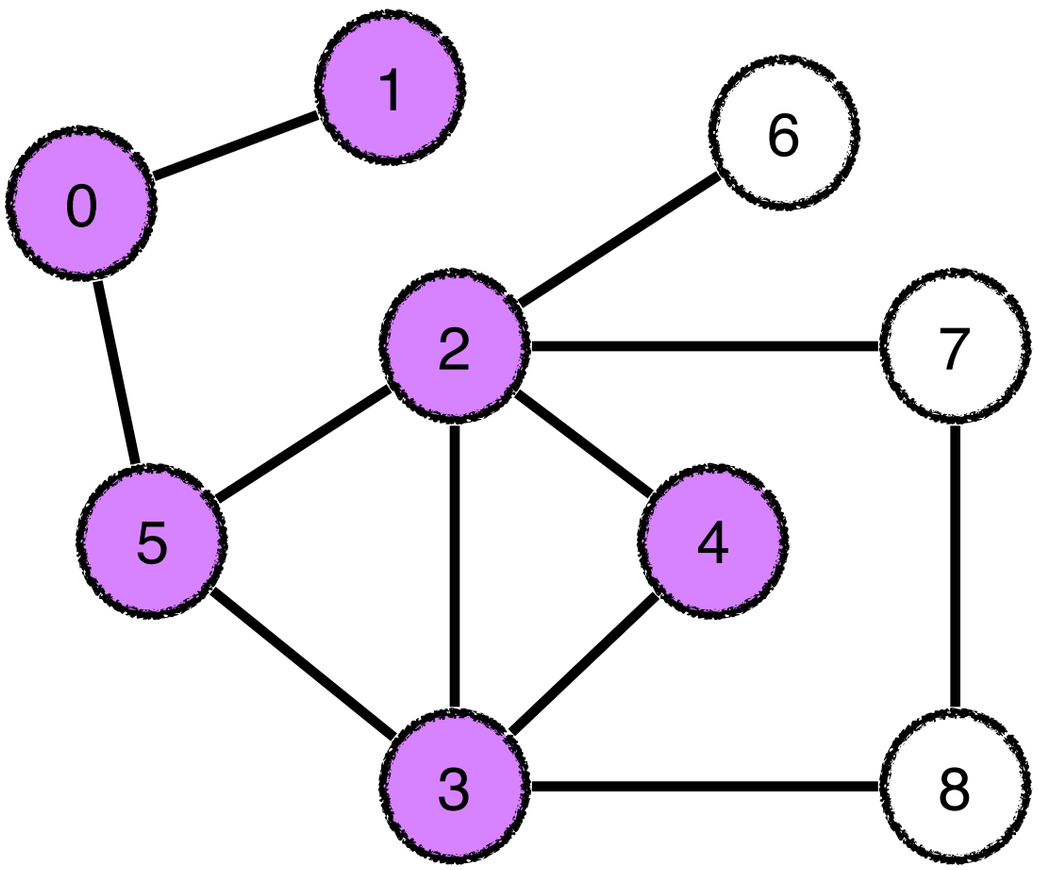
BFS





Added: 0 1 5 2 3

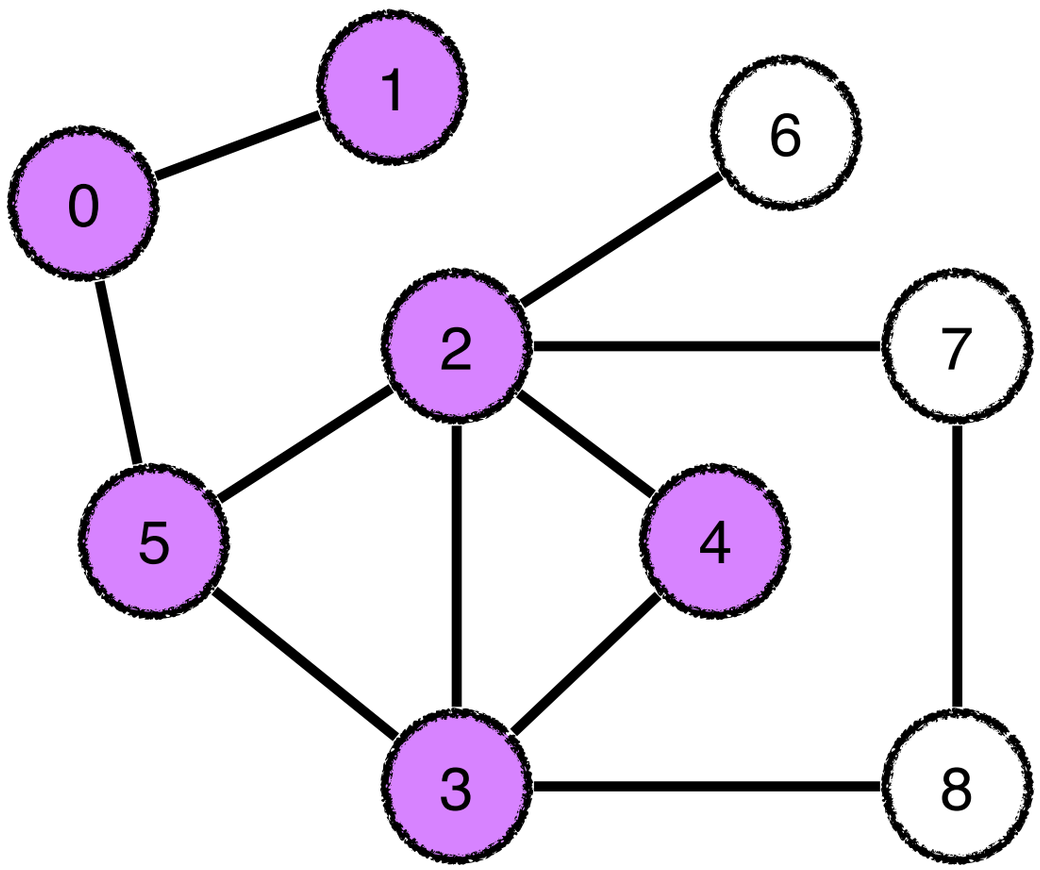
BFS



2
3
4

Added: 0 1 5 2 3

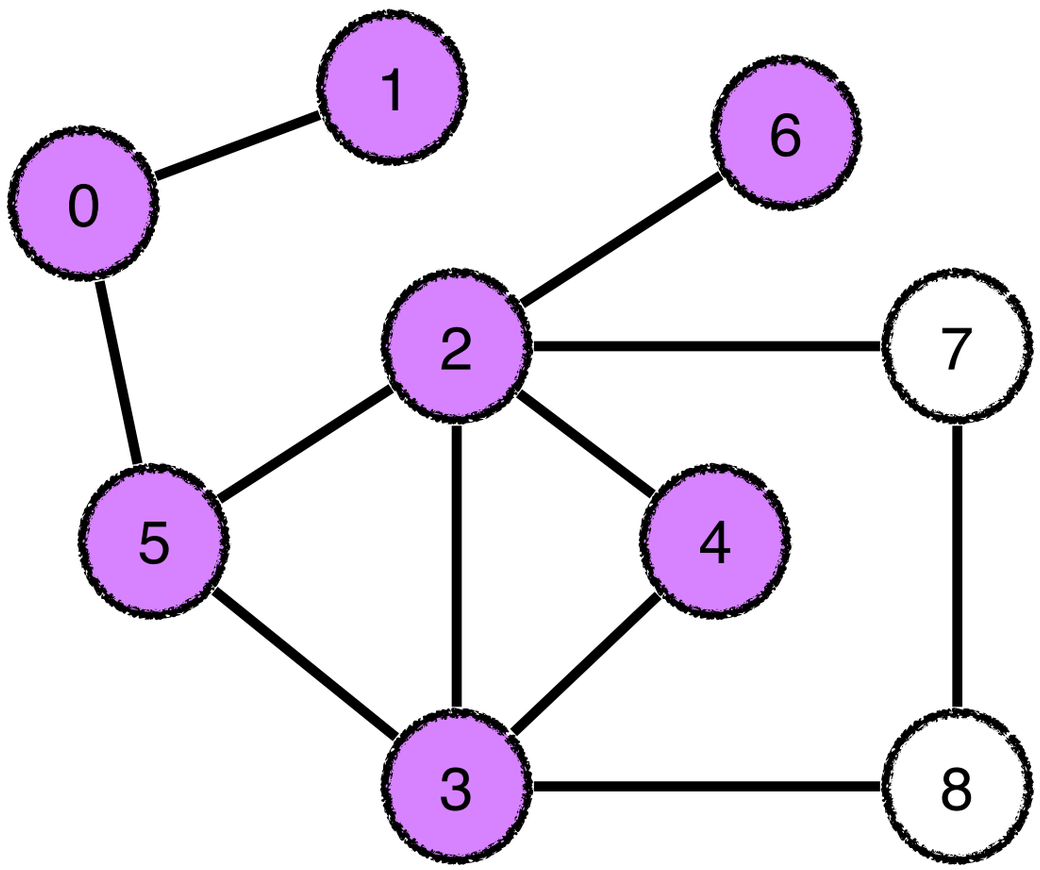
BFS



2
3
4

Added: 0 1 5 2 3 4

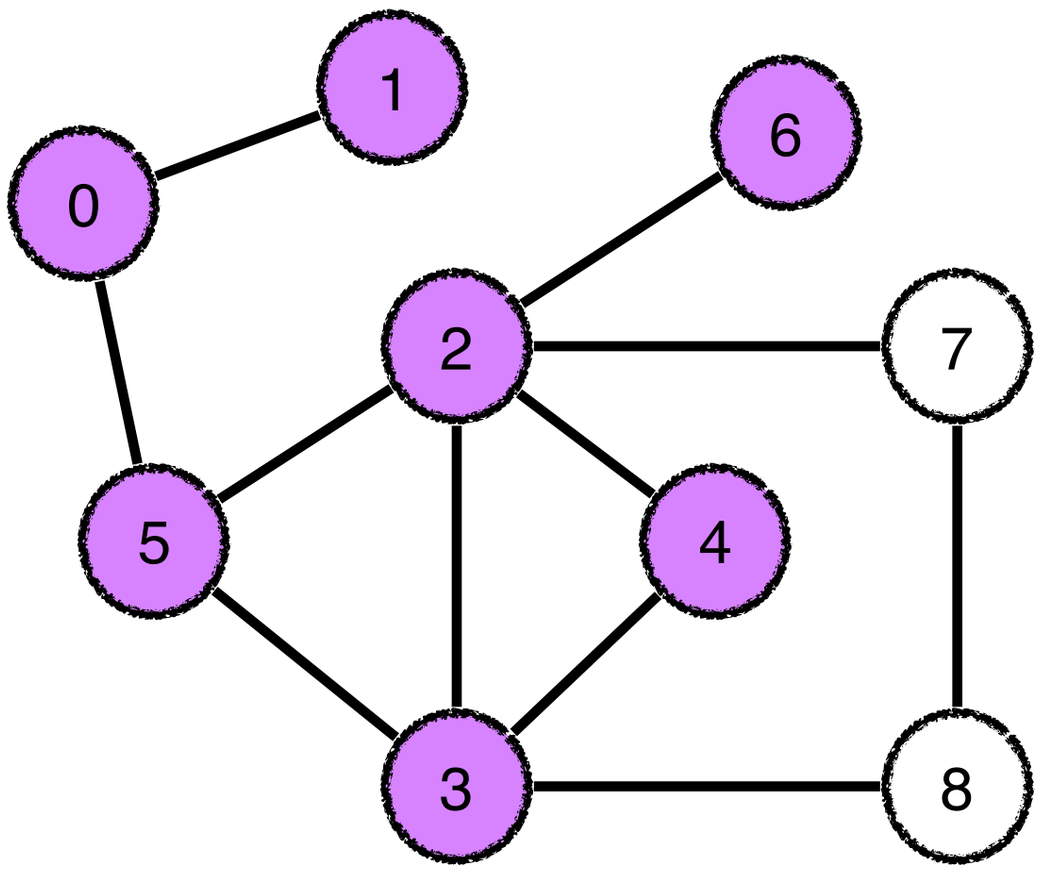
BFS

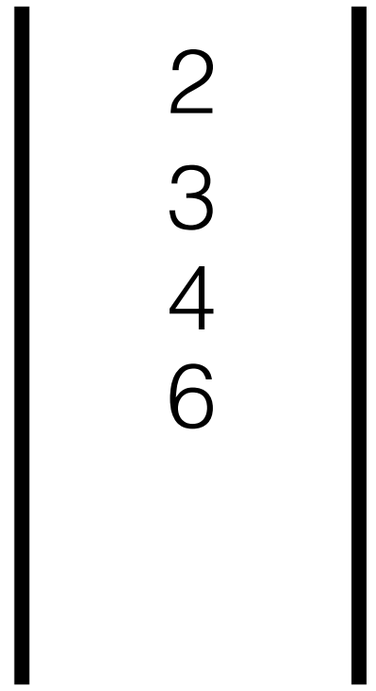


2
3
4

Added: 0 1 5 2 3 4

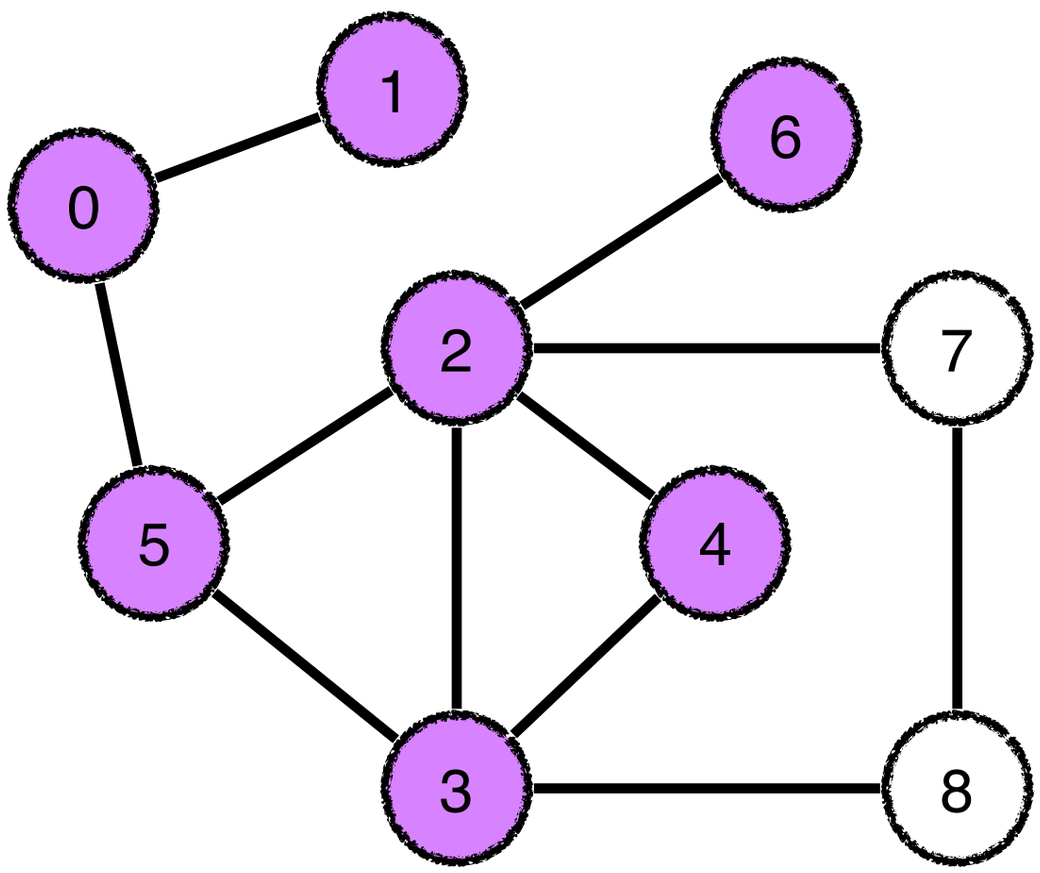
BFS

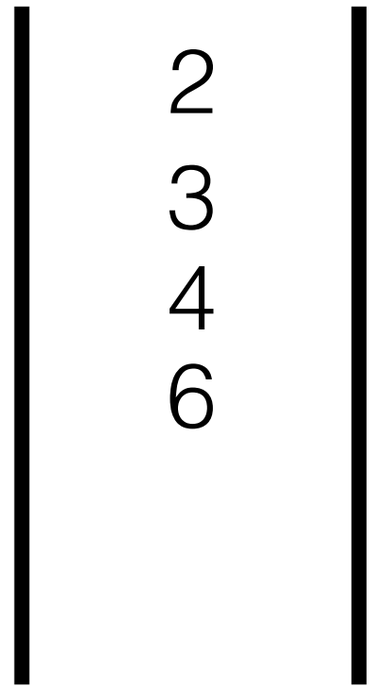




Added: 0 1 5 2 3 4

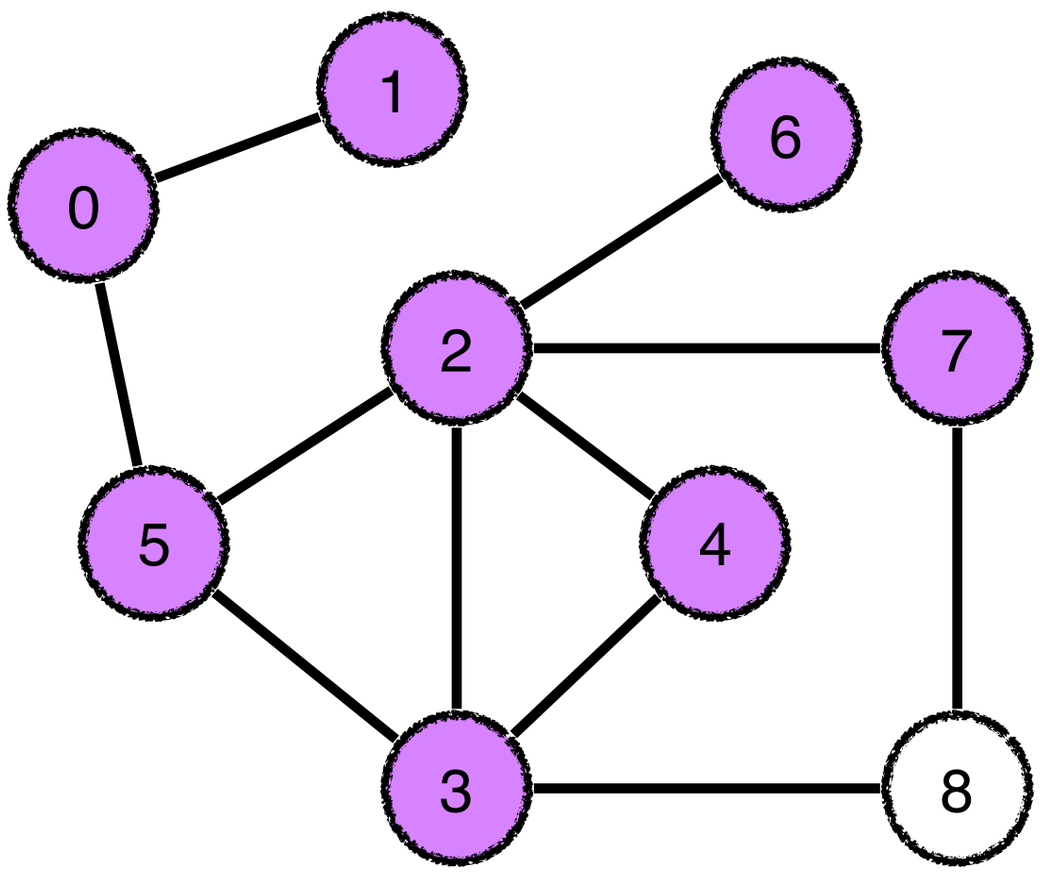
BFS

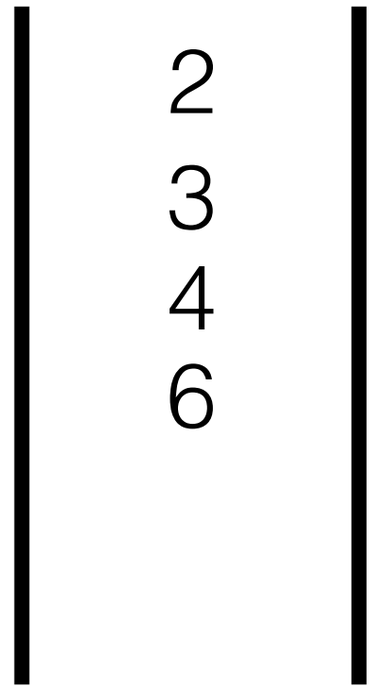




Added: 0 1 5 2 3 4 6

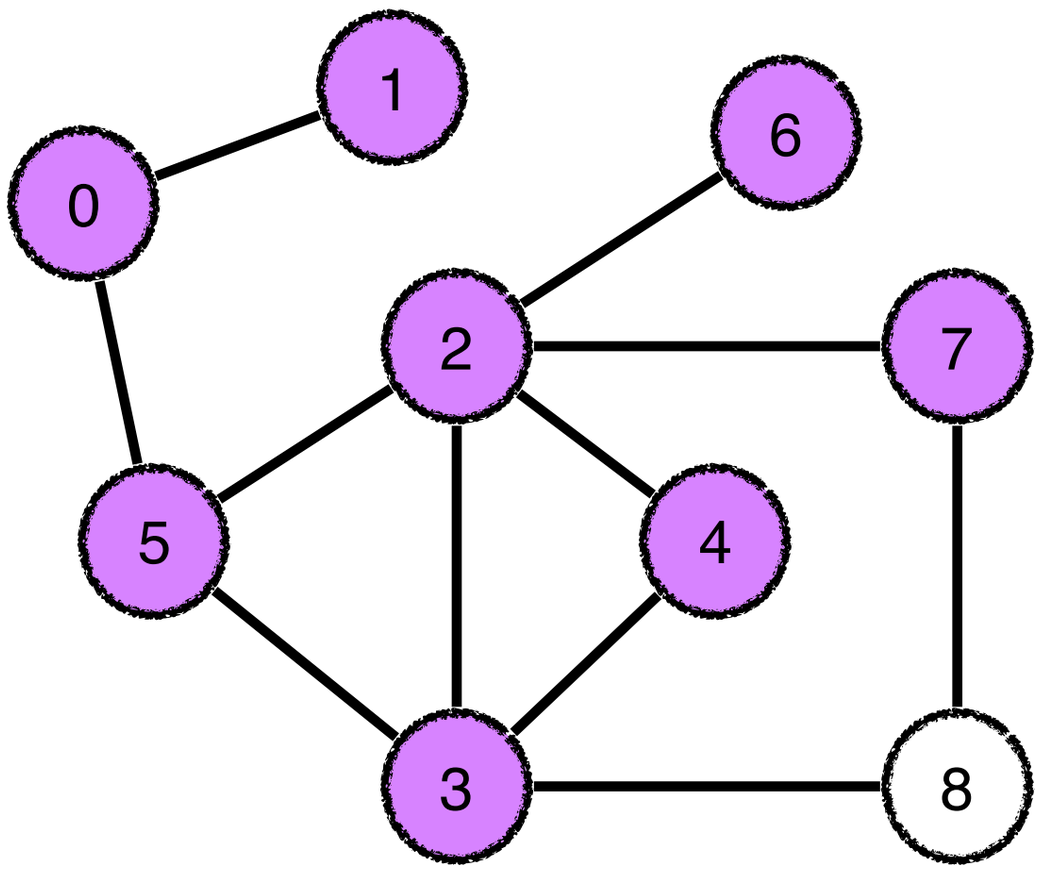
BFS





Added: 0 1 5 2 3 4 6

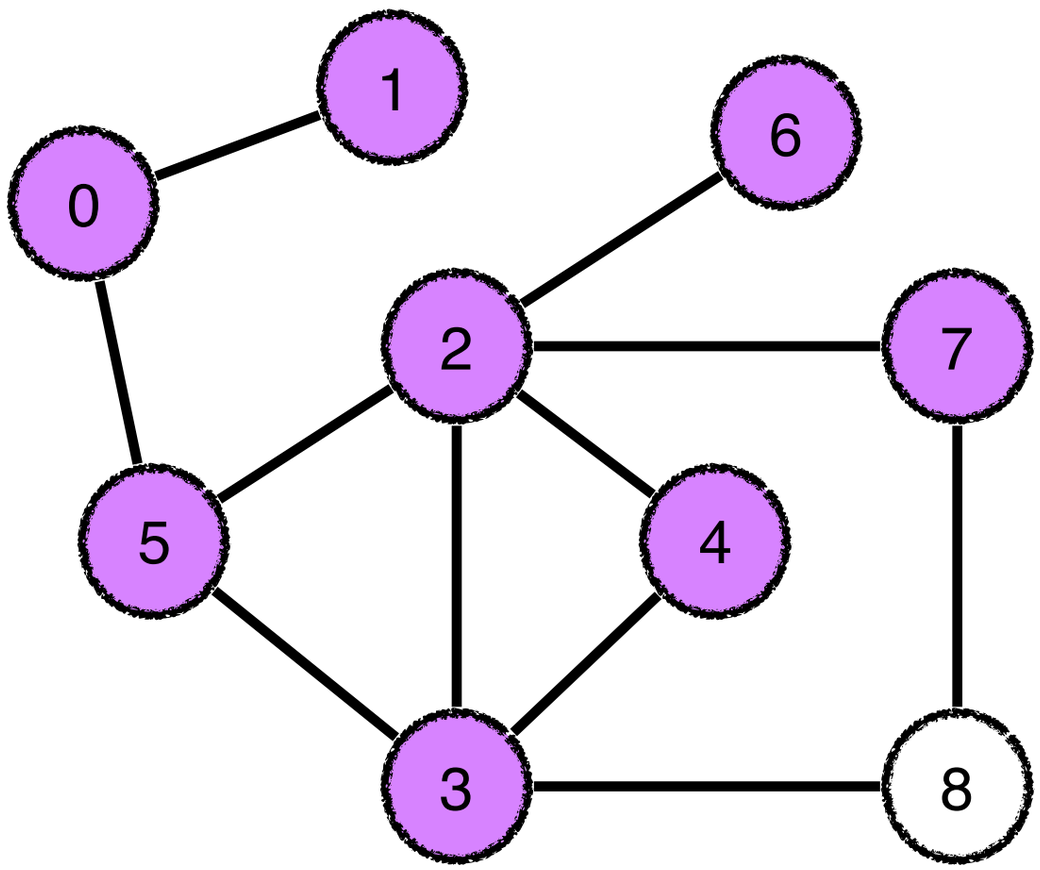
BFS



2
3
4
6
7

Added: 0 1 5 2 3 4 6

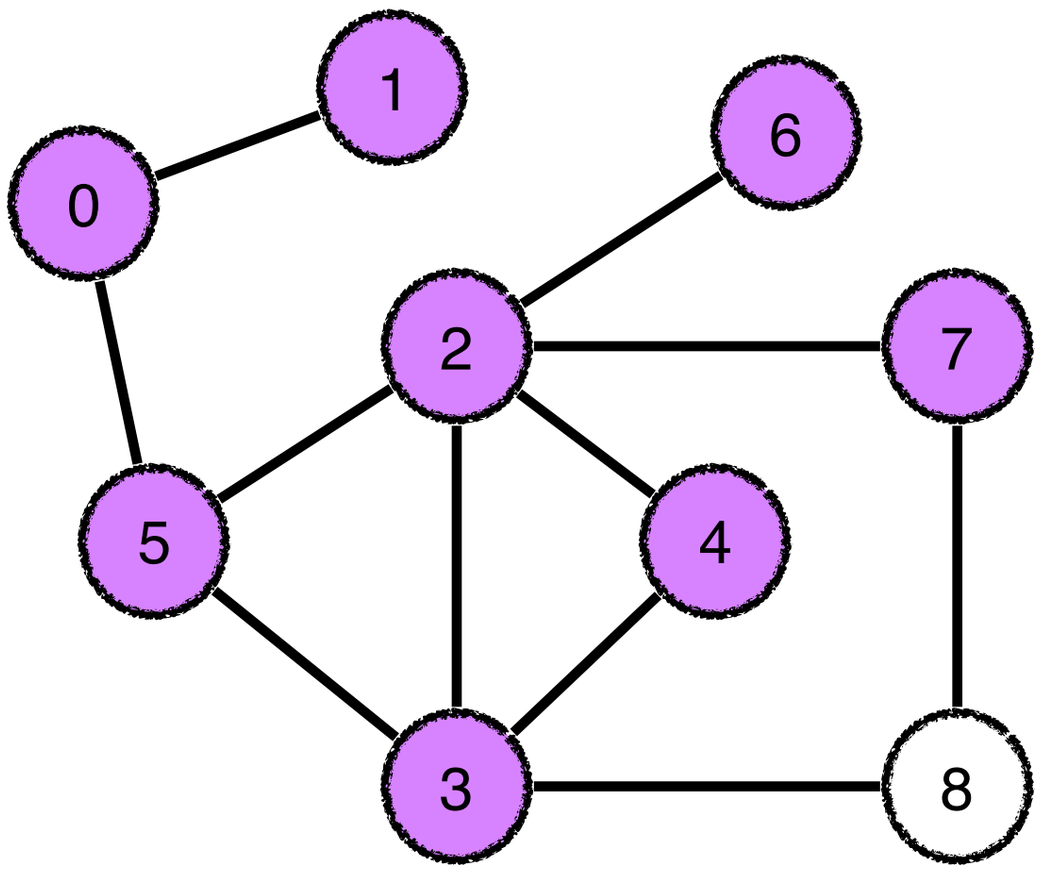
BFS

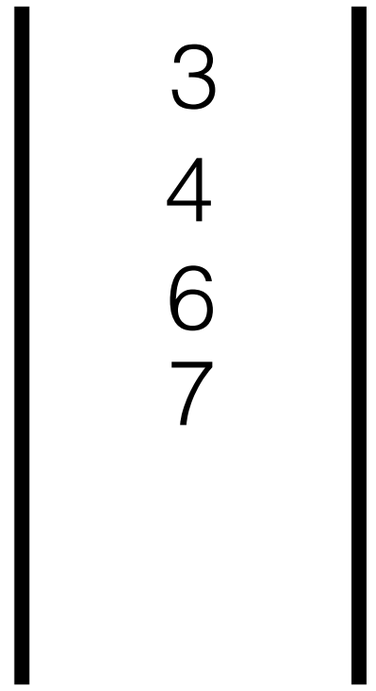


2
3
4
6
7

Added: 0 1 5 2 3 4 6 7

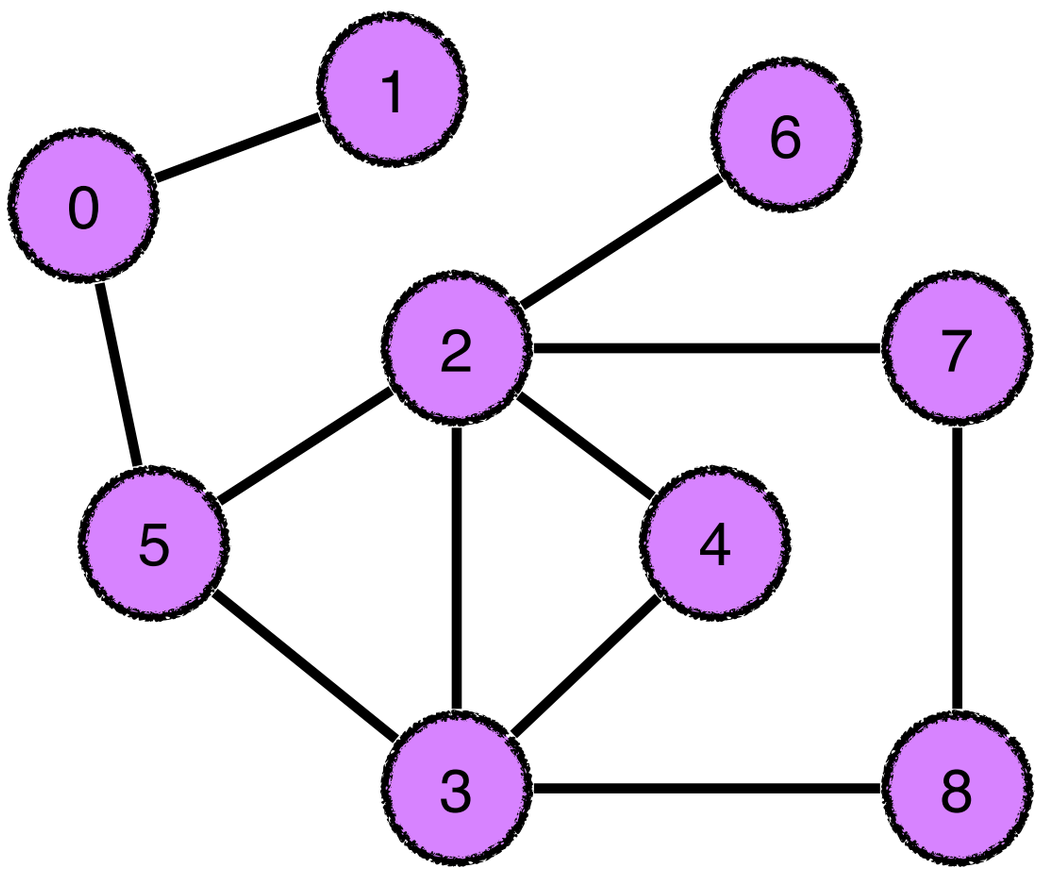
BFS





Added: 0 1 5 2 3 4 6 7

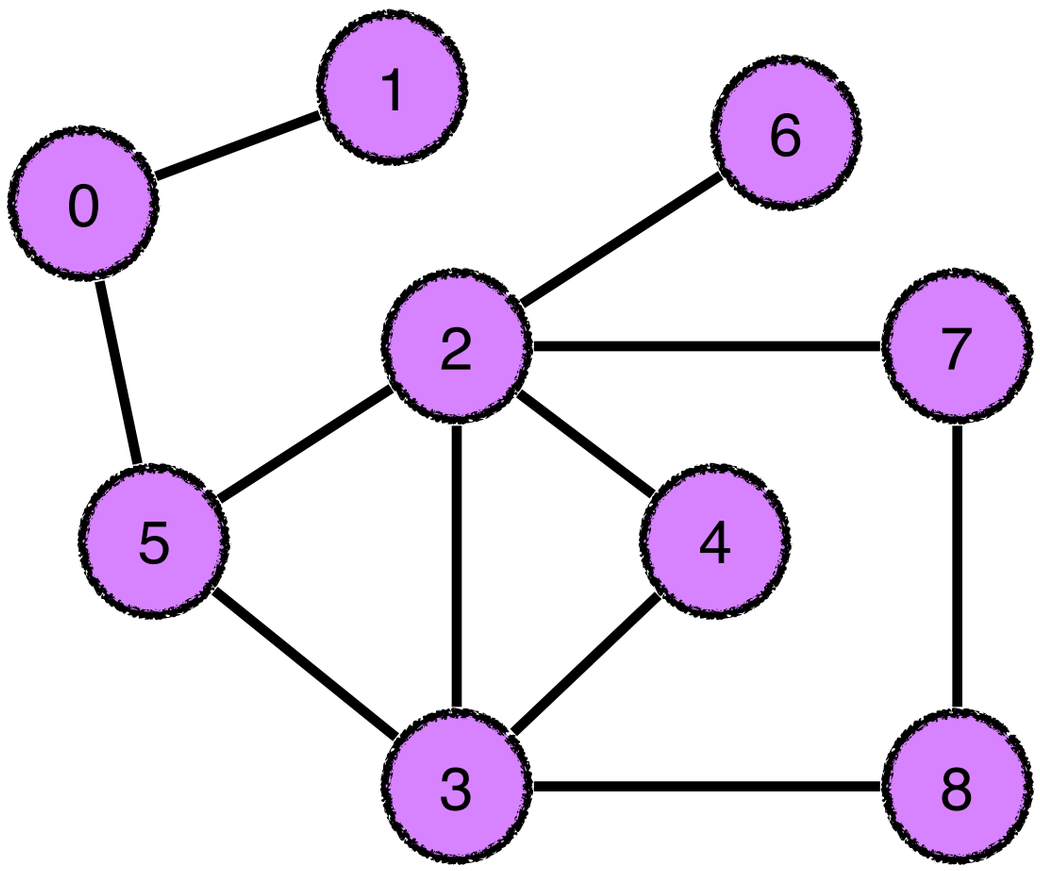
BFS

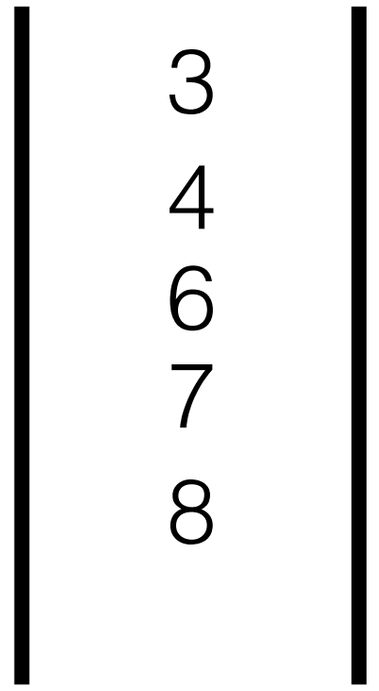


3
4
6
7

Added: 0 1 5 2 3 4 6 7

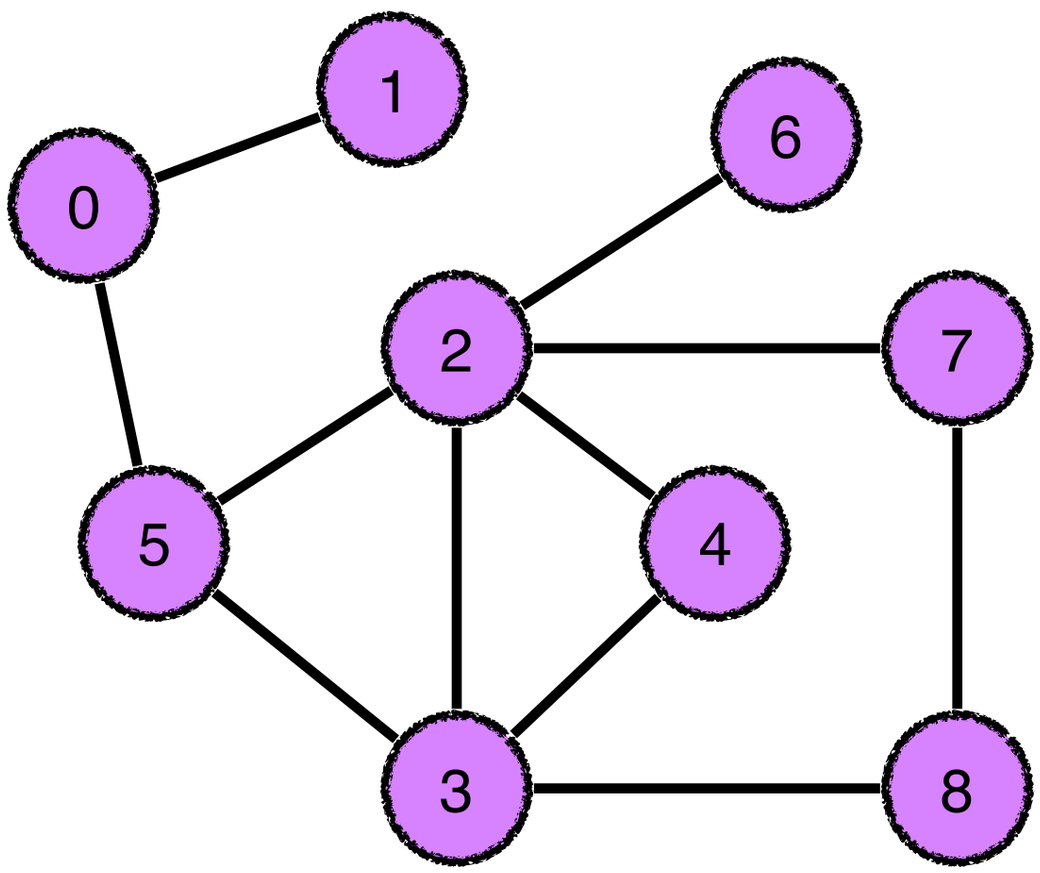
BFS





Added: 0 1 5 2 3 4 6 7

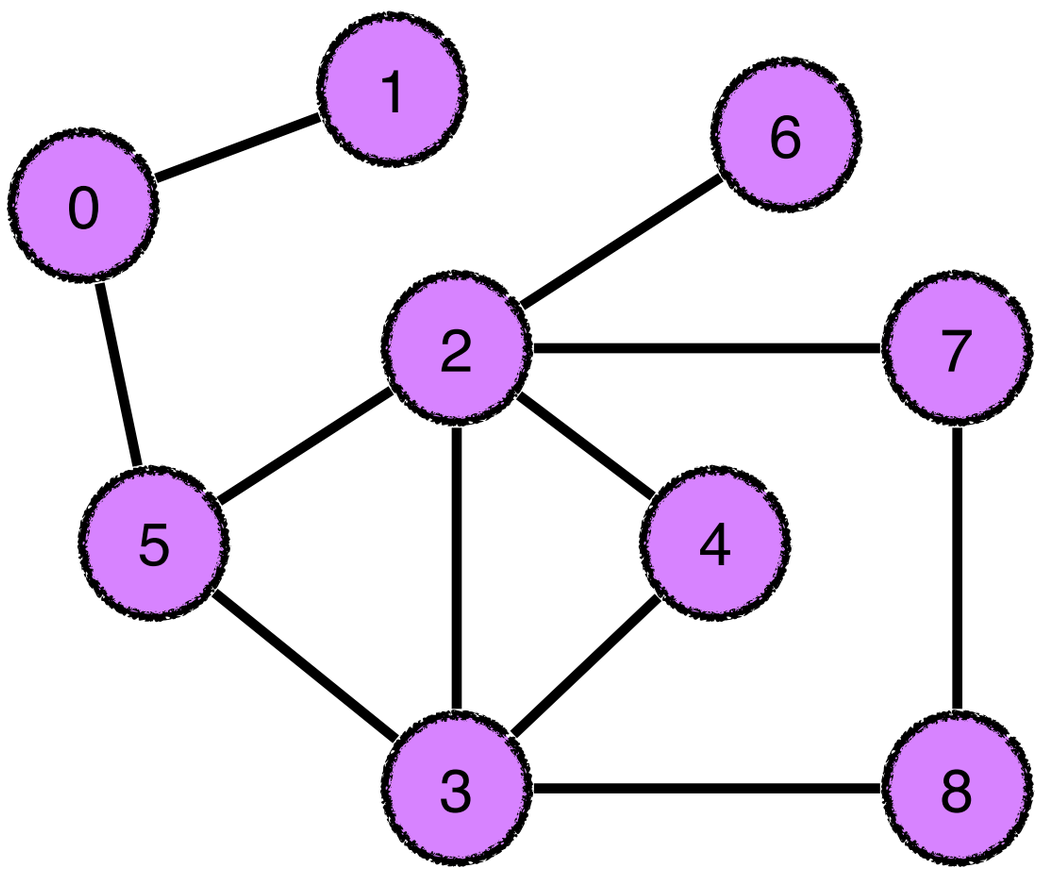
BFS

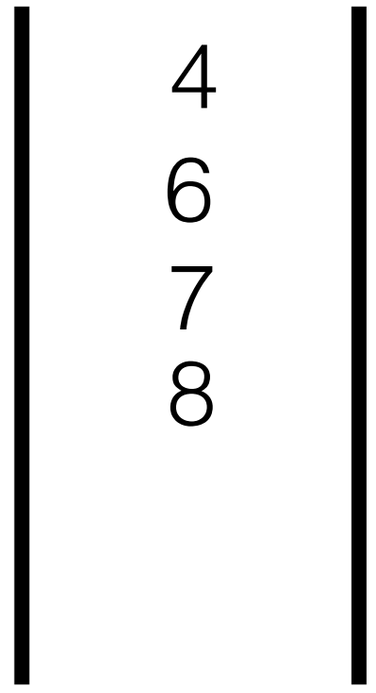


3
4
6
7
8

Added: 0 1 5 2 3 4 6 7 8

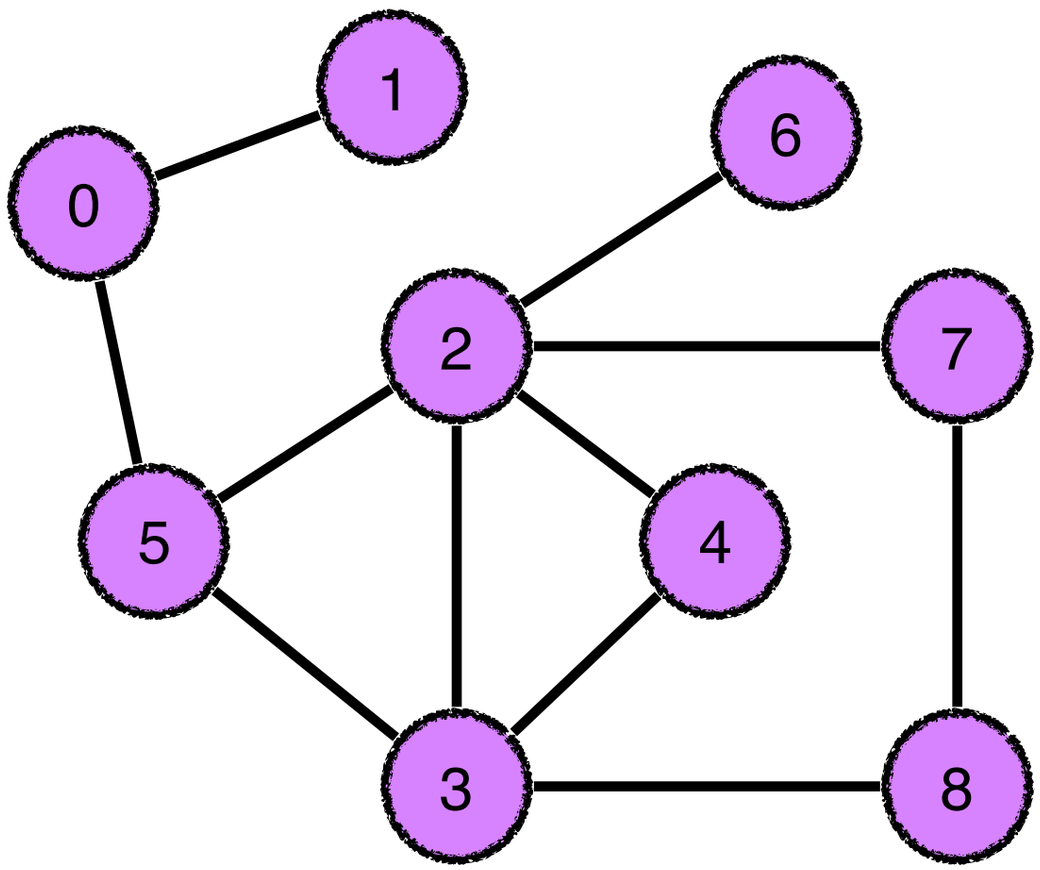
BFS

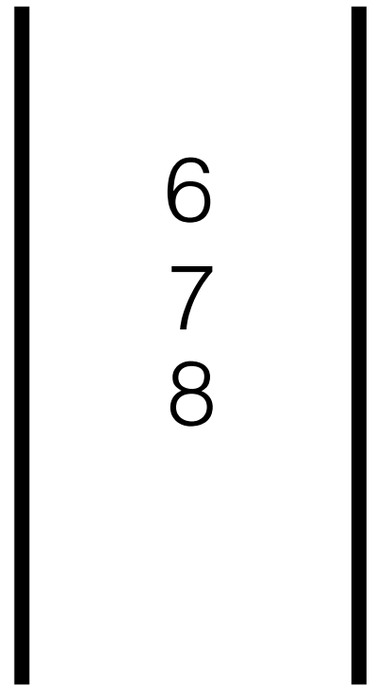




Added: 0 1 5 2 3 4 6 7 8

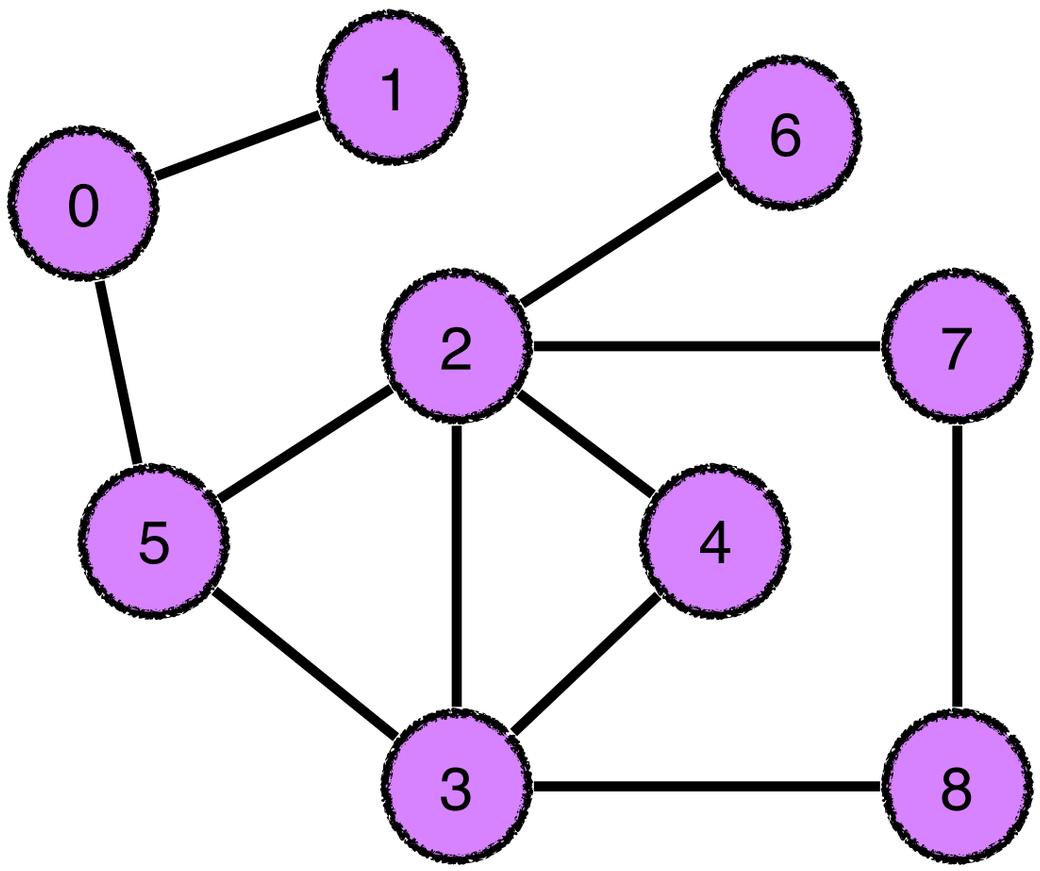
BFS

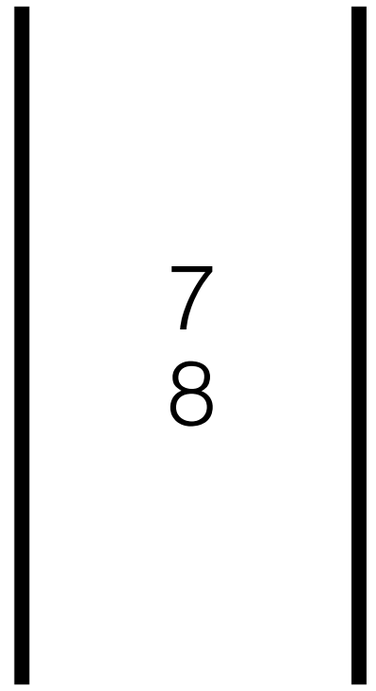




Added: 0 1 5 2 3 4 6 7 8

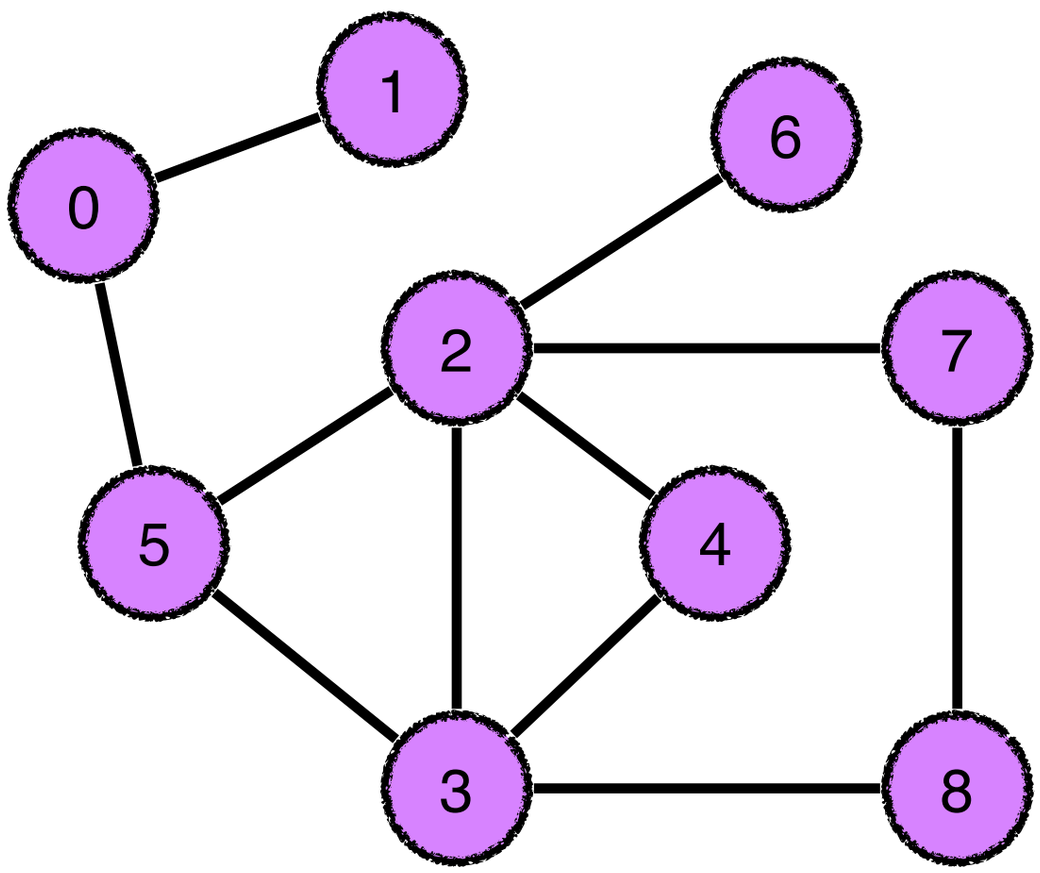
BFS

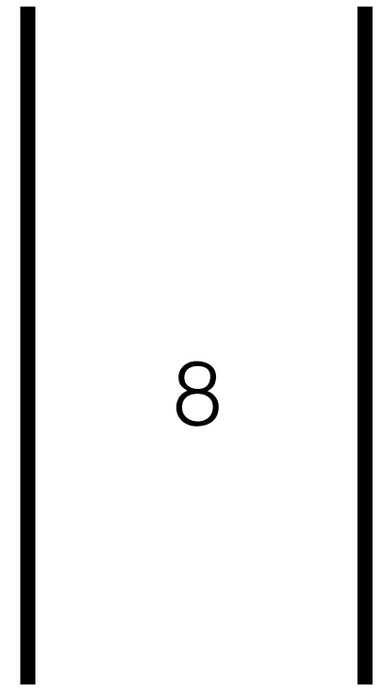




Added: 0 1 5 2 3 4 6 7 8

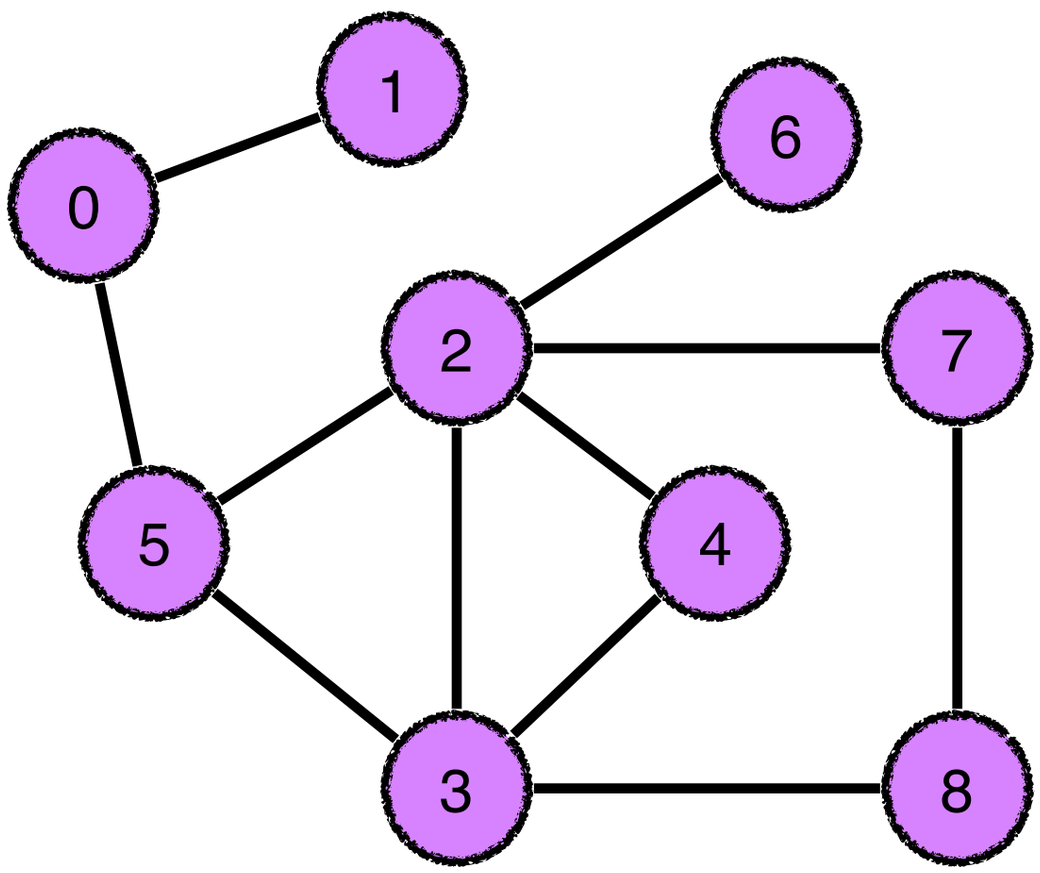
BFS





Added: 0 1 5 2 3 4 6 7 8

BFS



Added: 0 1 5 2 3 4 6 7 8

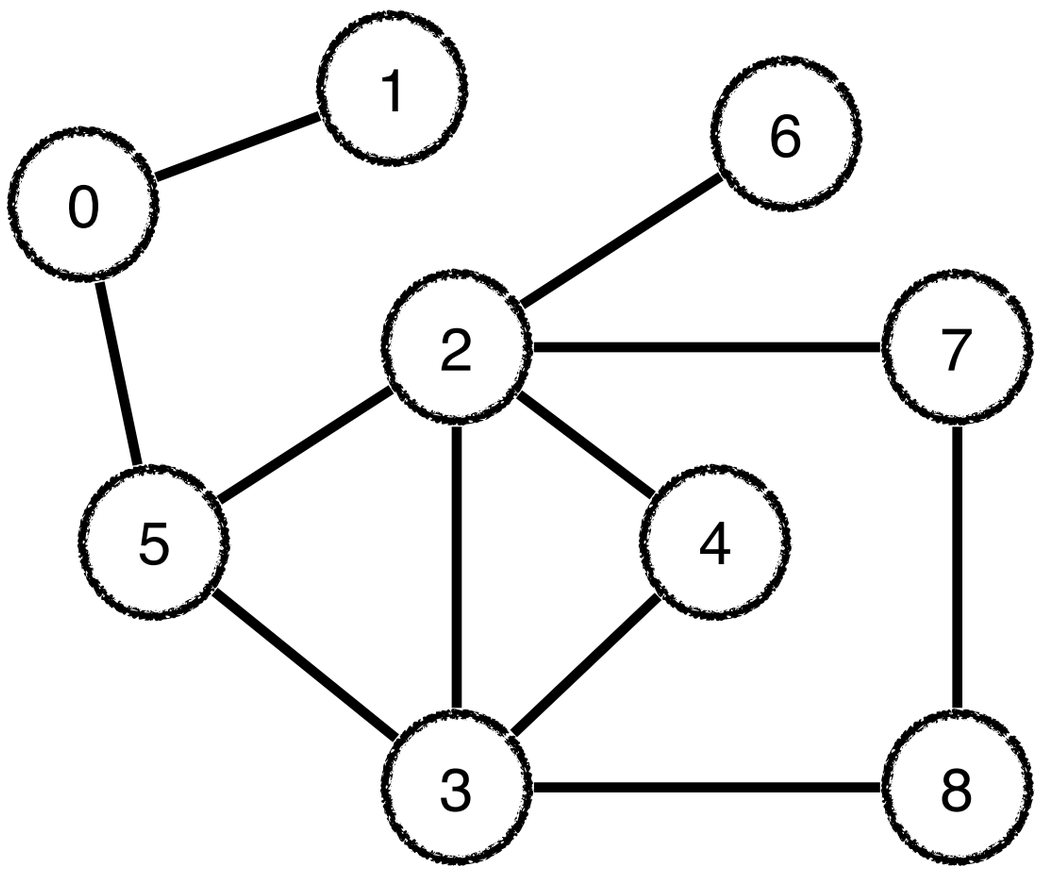
BFS

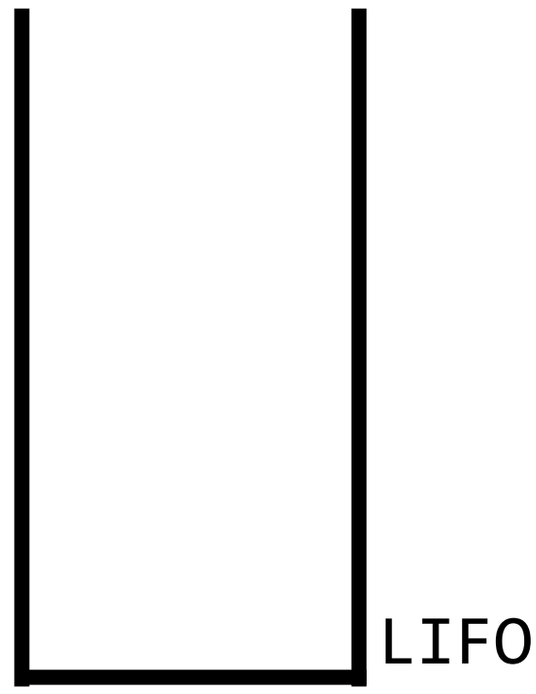
```
fun bfs(nodes : Array<List<Int>>, source : Int, n : Int):Array<Int> {
    var parent = Array(n) { x -> -1}
    var fifo = Array(n) { x -> -1 }
    var visited = Array(n) { x -> 0}
    var front = -1
    var rear = -1

    fifo[++rear] = source
    var k : Int
    while (front != rear) {
        k = fifo[++front]
        var list = nodes.get(k)
        for (v in list) {
            if (visited[v] == 0) {
                fifo[++rear] = v
                visited[v] = 1
                parent[v] = k
            }
        }
    }
    return parent
}
```

concat to list

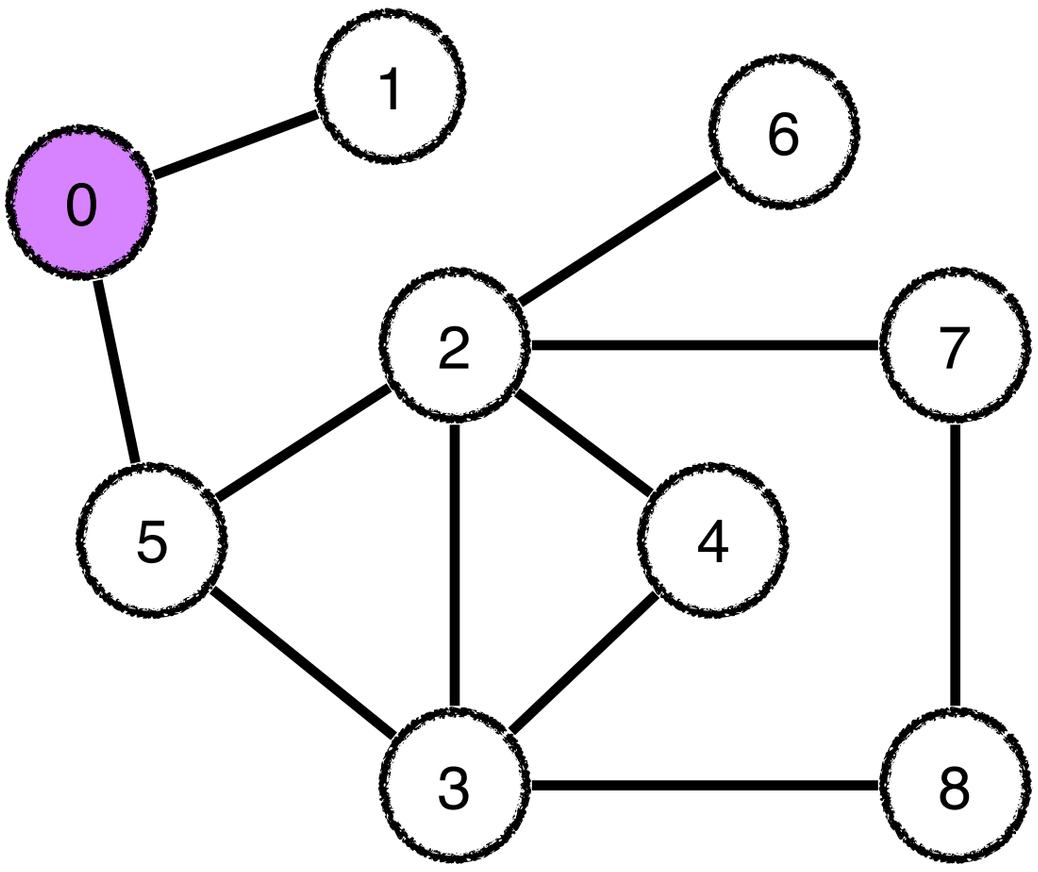
cdr the list

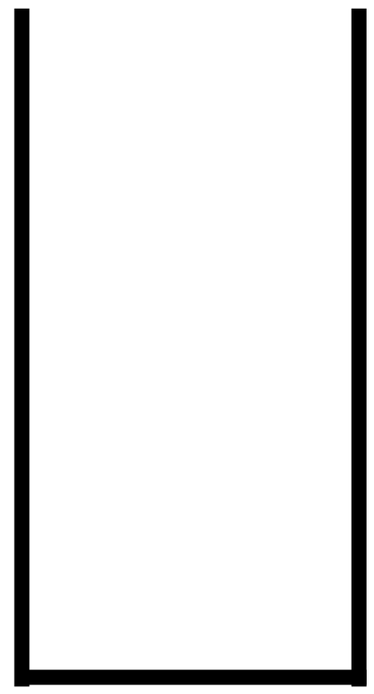
Depth-first search





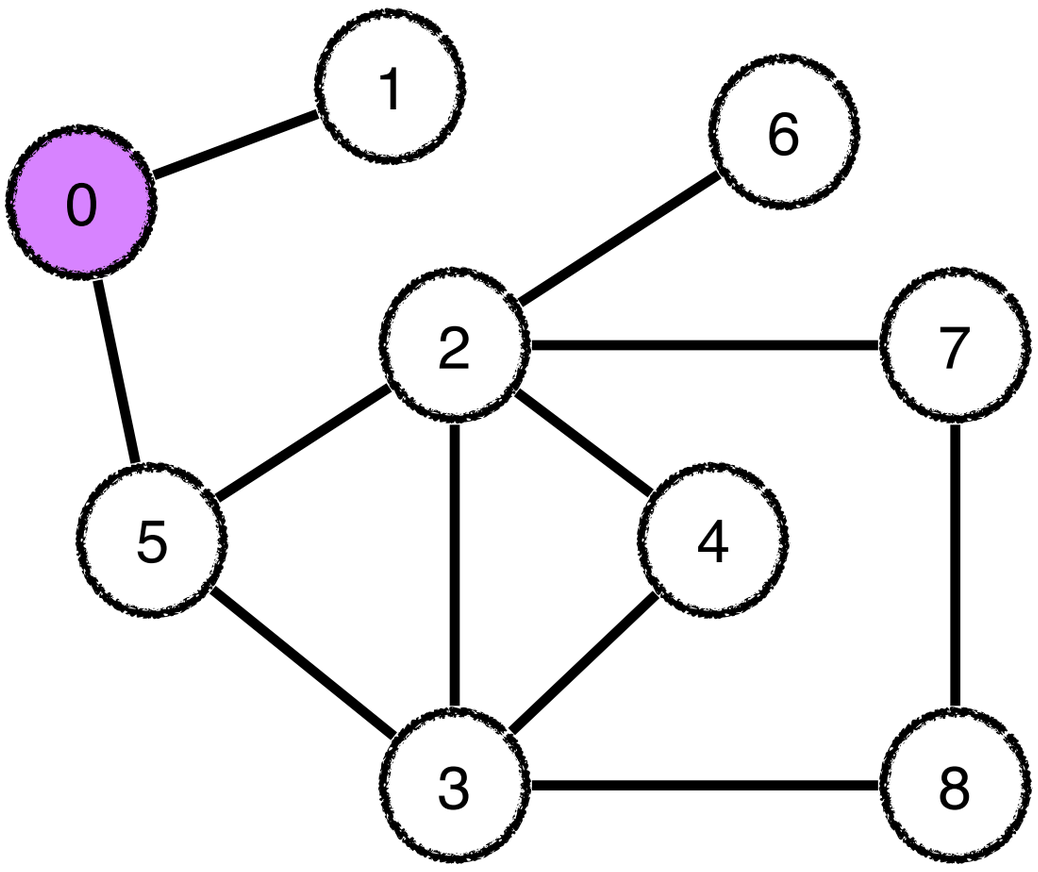
DFS

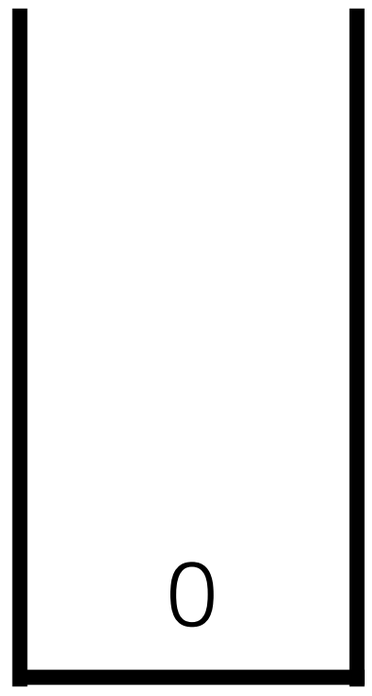




Added:

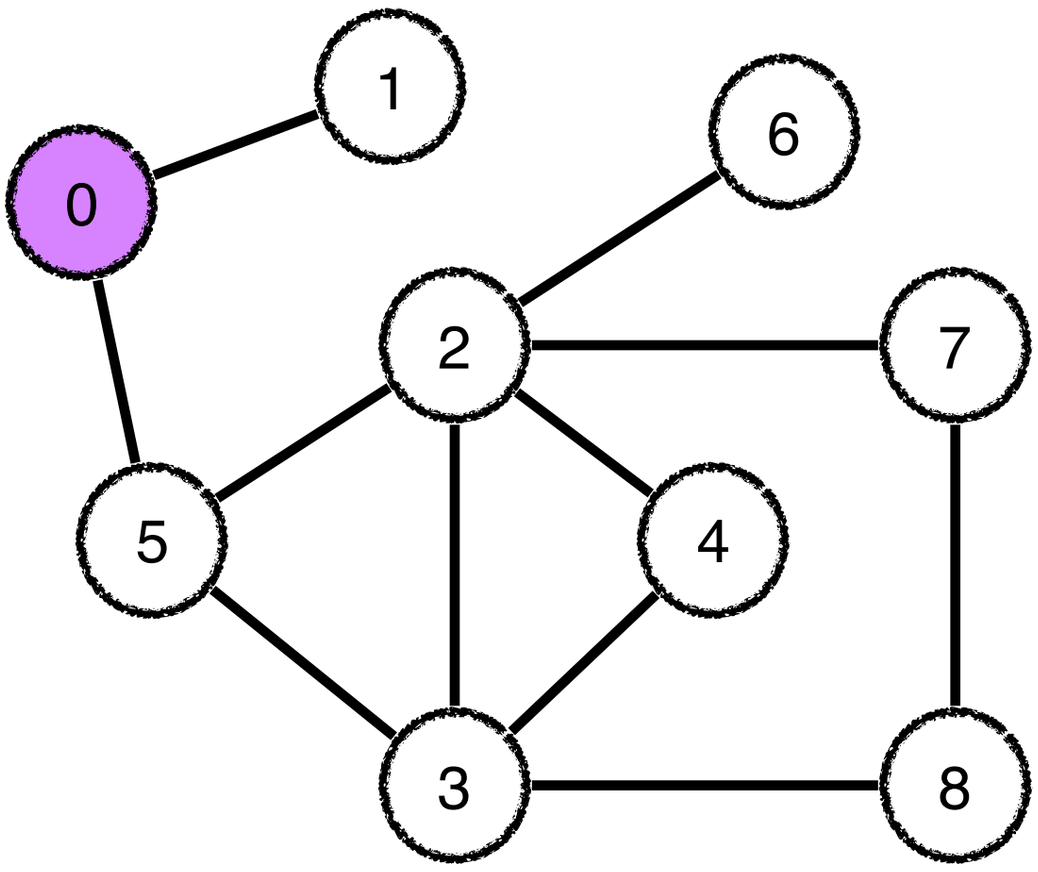
DFS

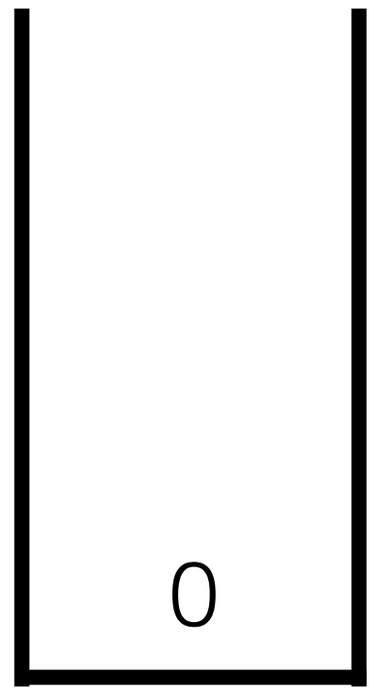




Added:

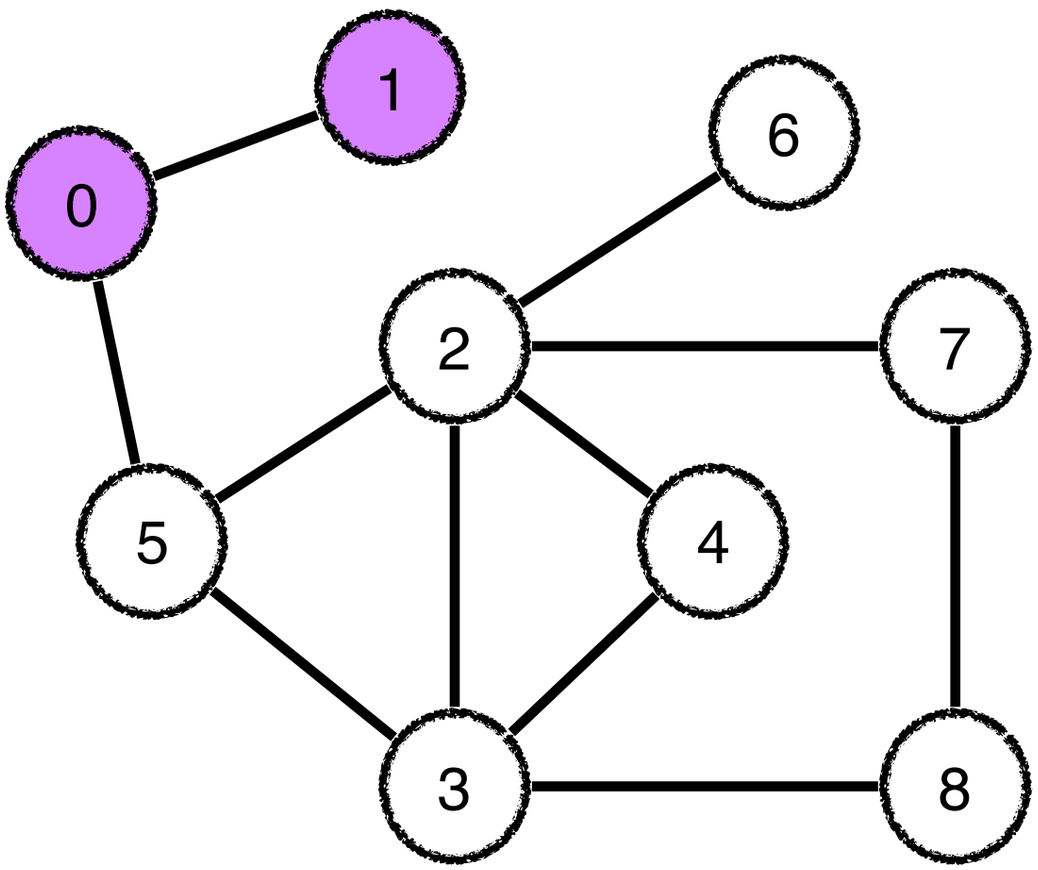
DFS

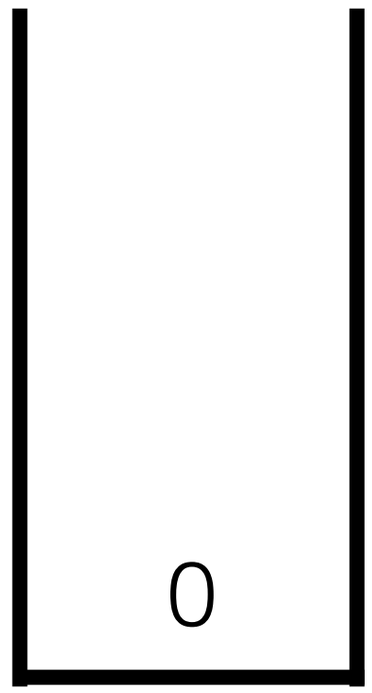




Added: 0

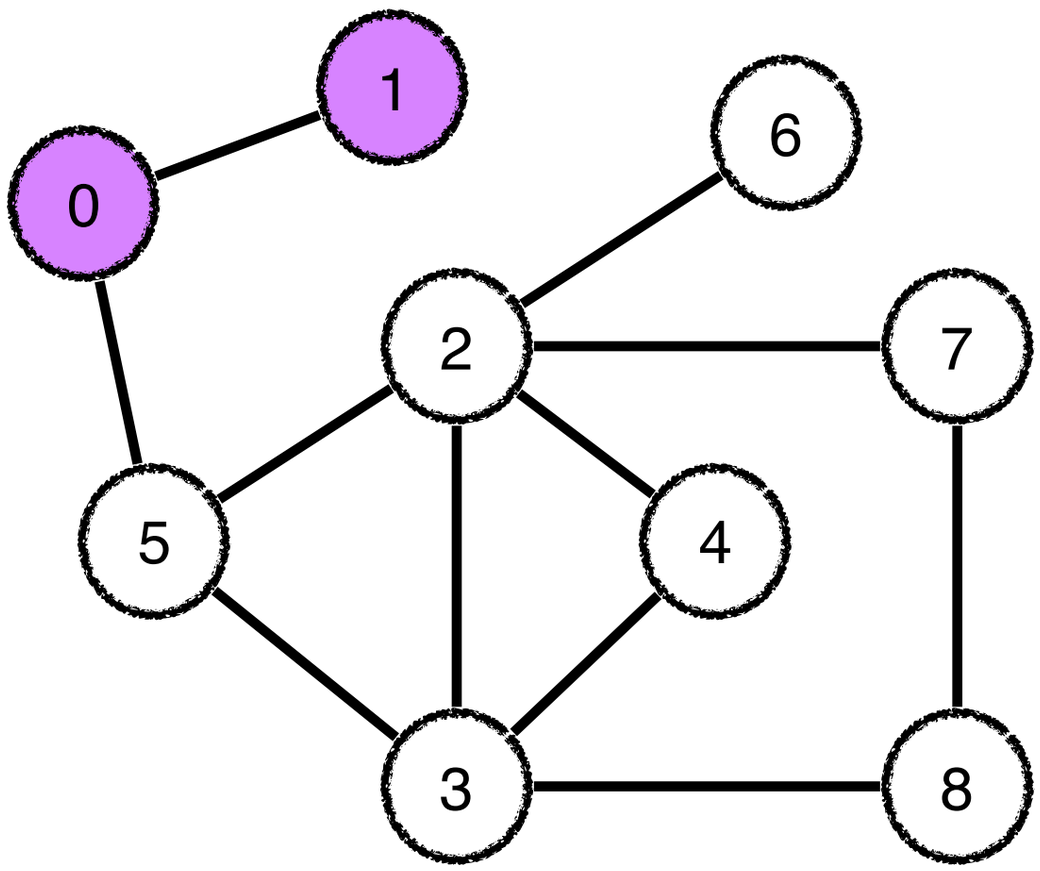
DFS

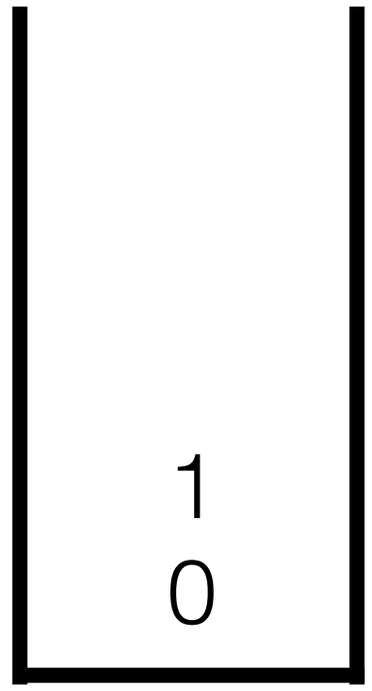




Added: 0

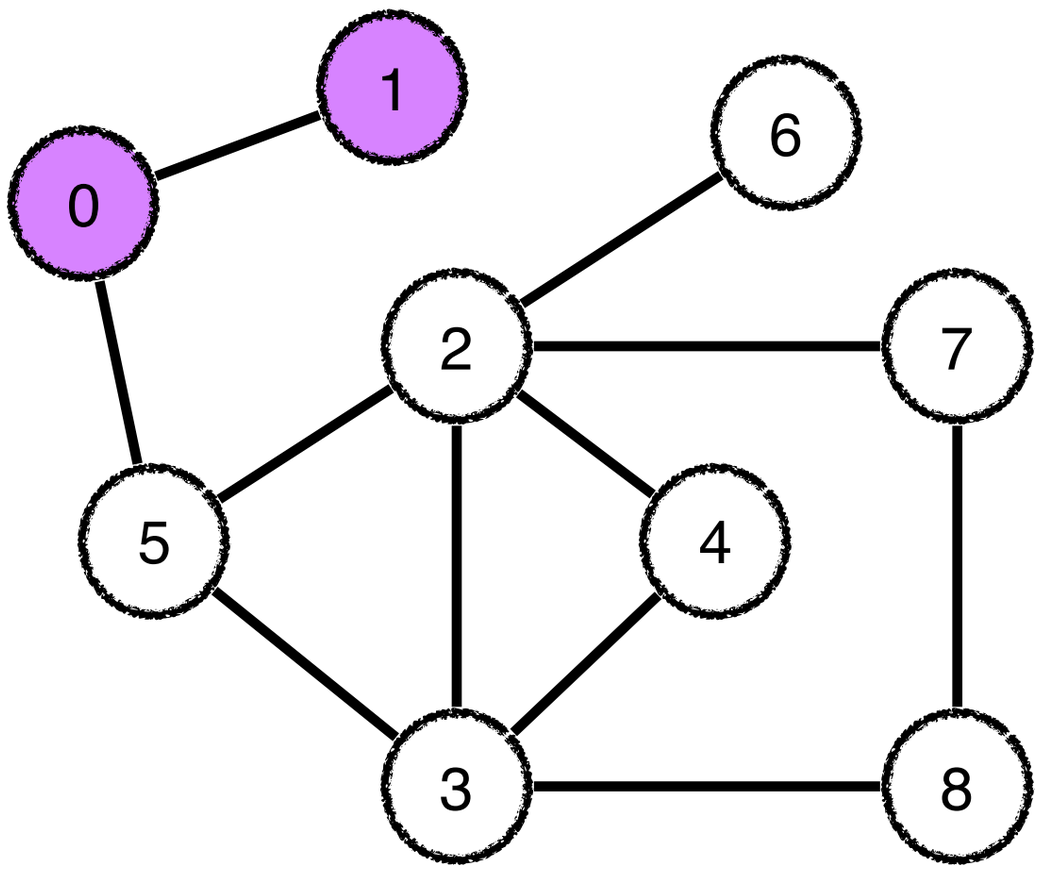
DFS

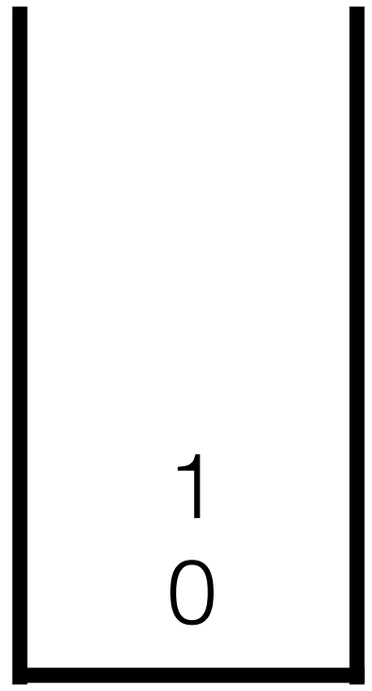




Added: 0

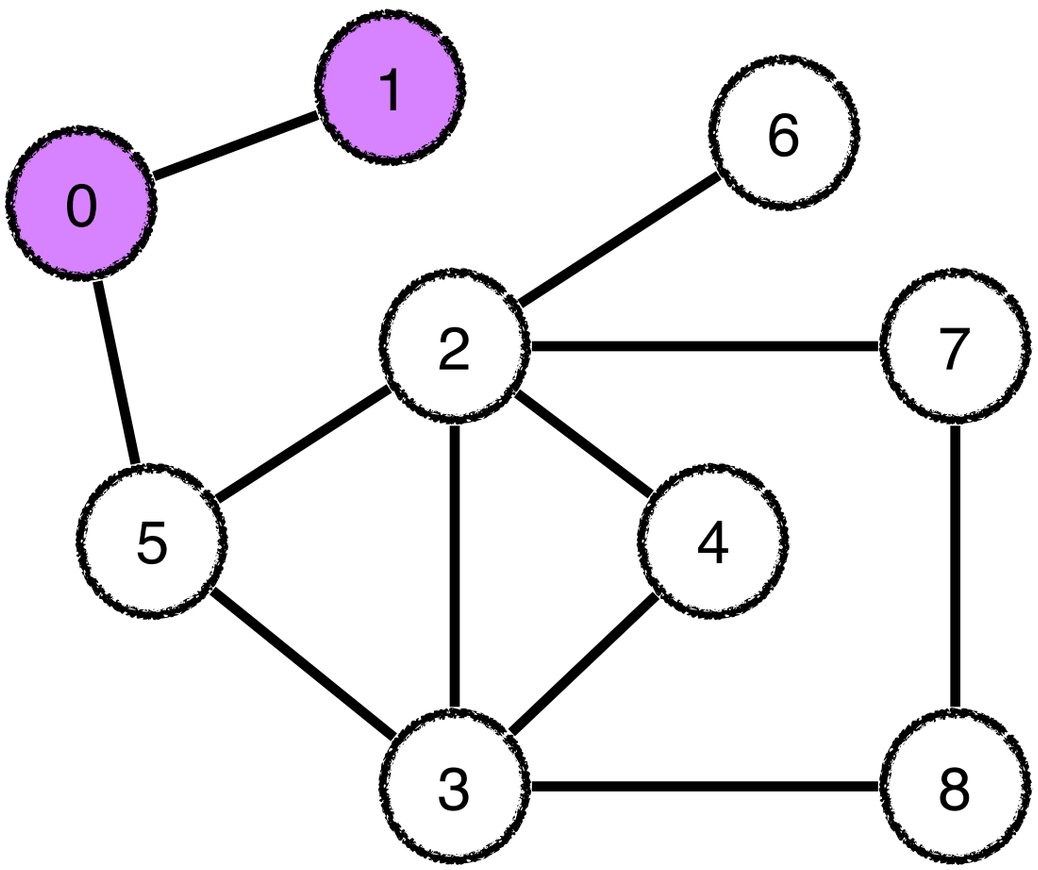
DFS

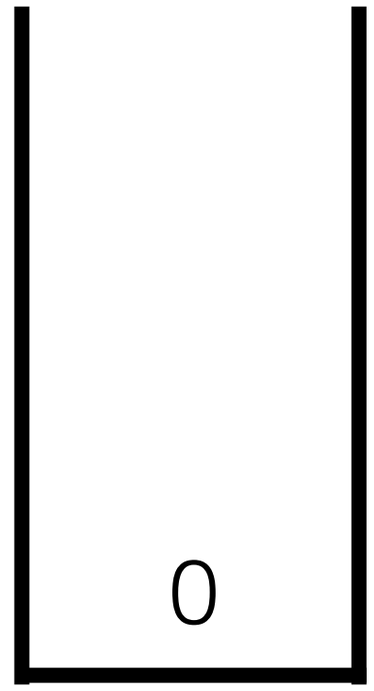




Added: 0 1

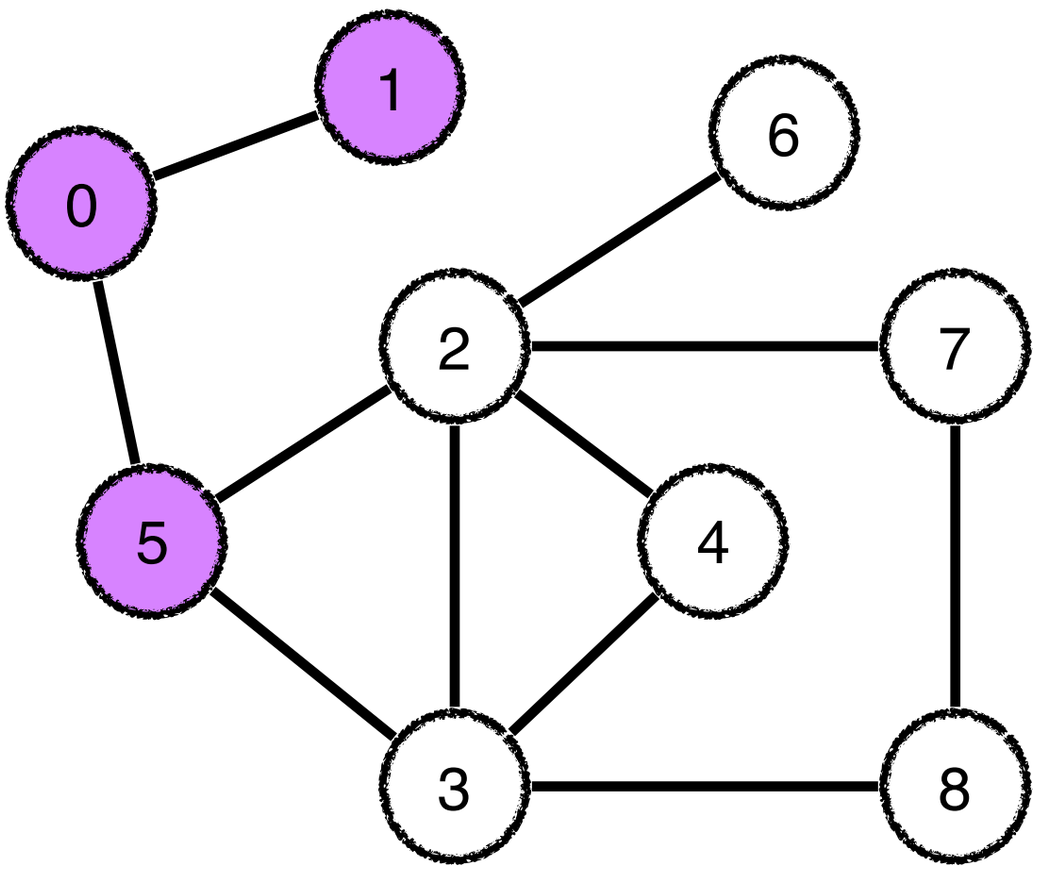
DFS

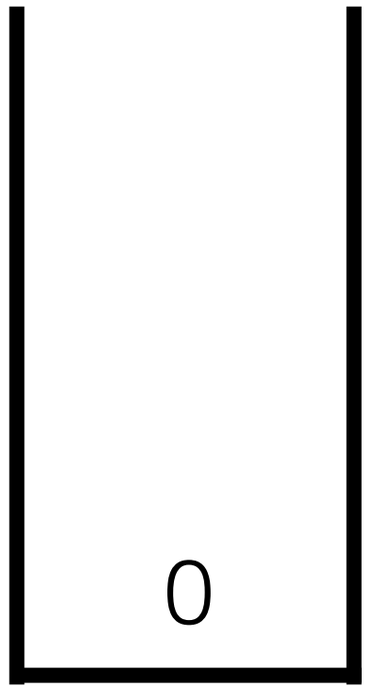




Added: 0 1

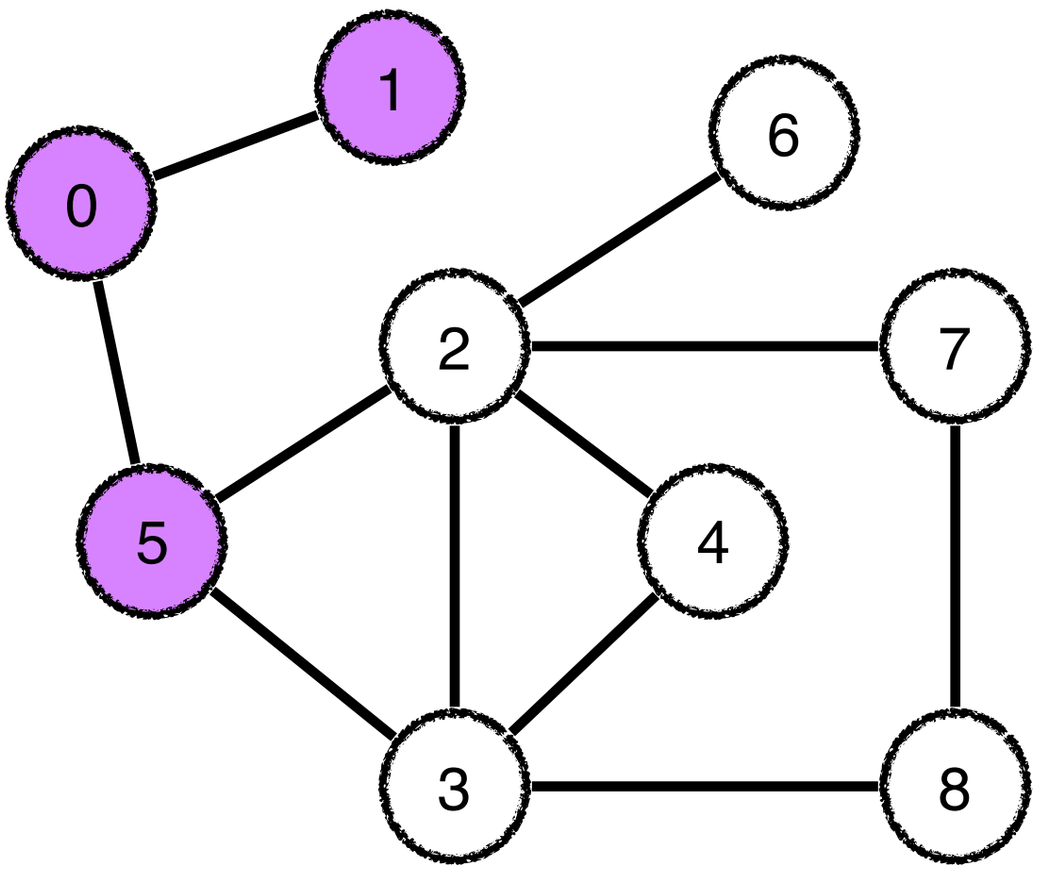
DFS

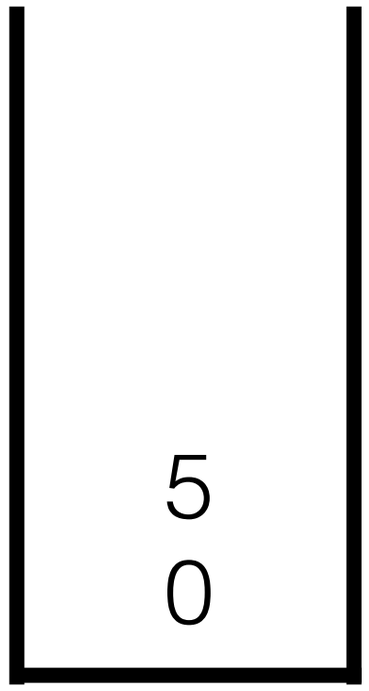




Added: 0 1

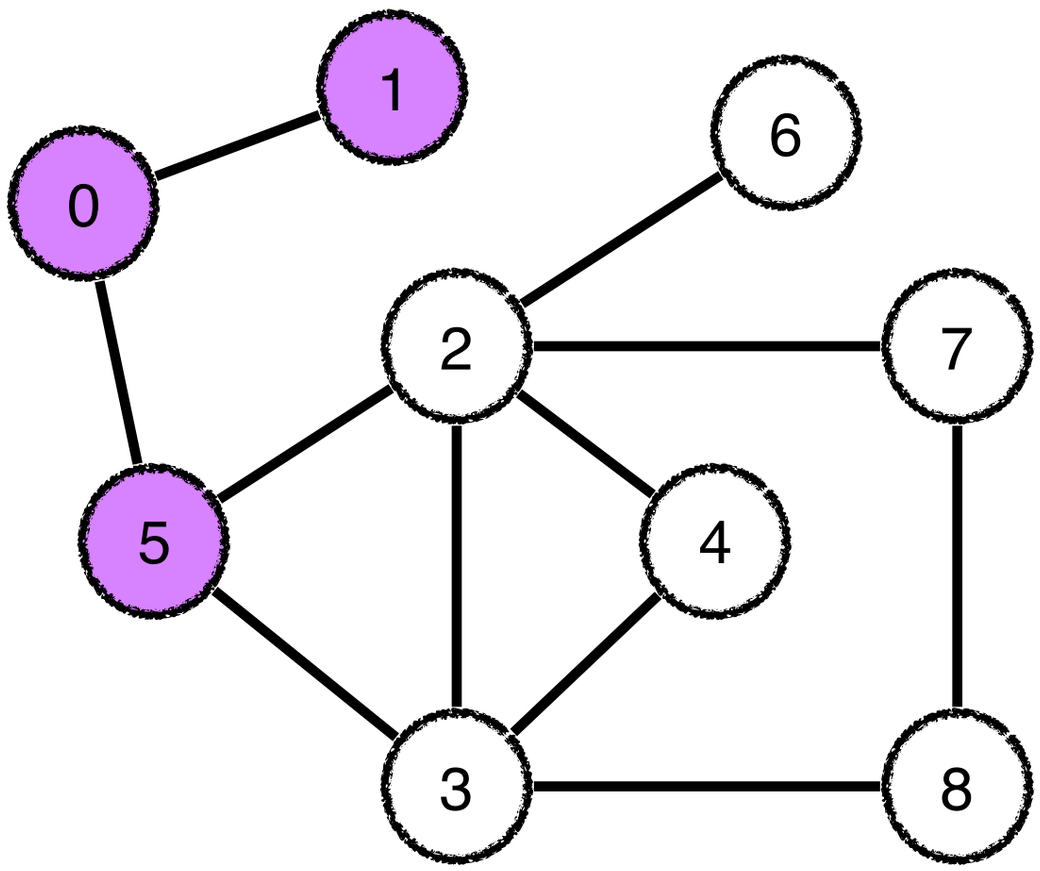
DFS

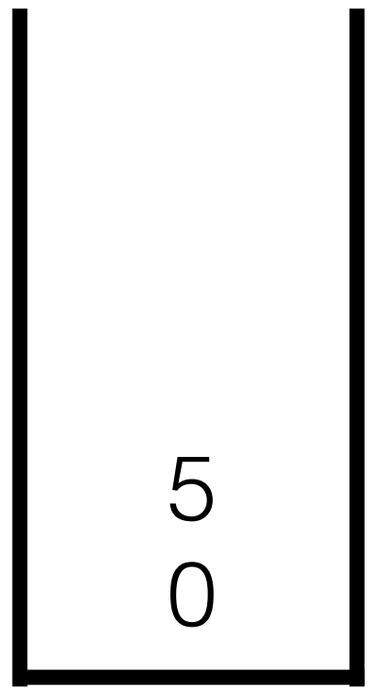




Added: 0 1

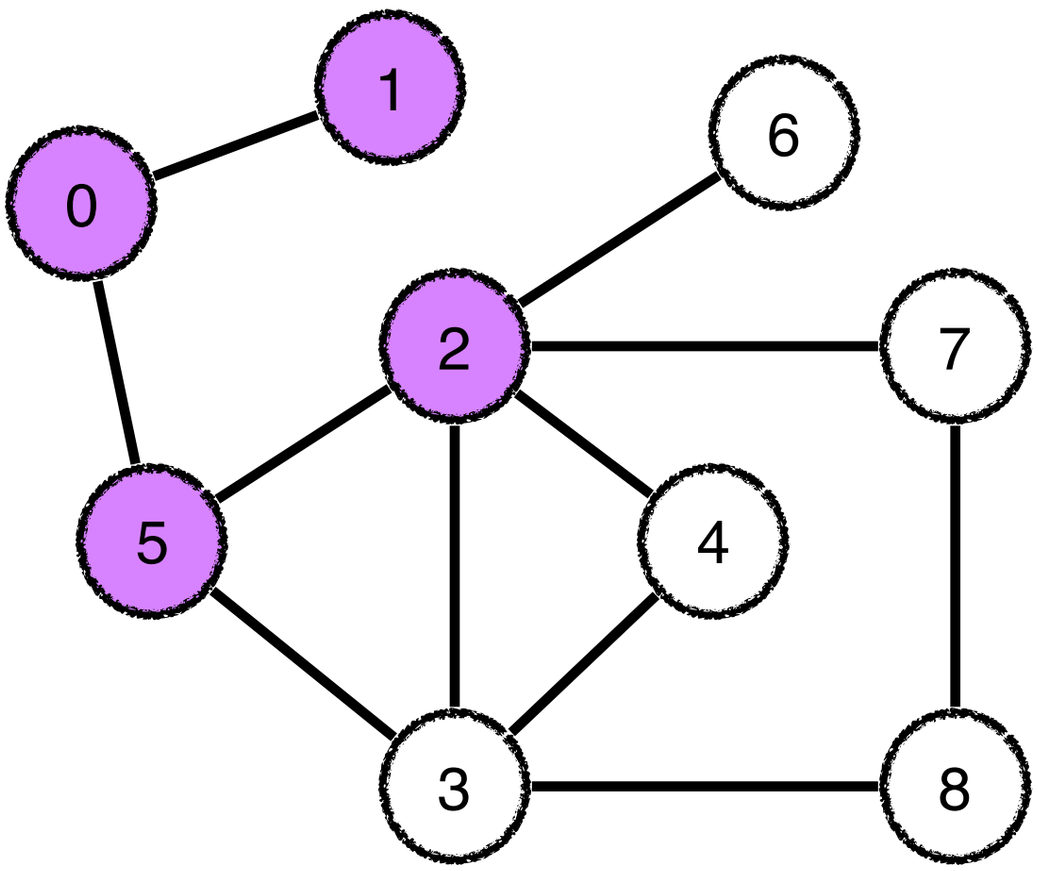
DFS

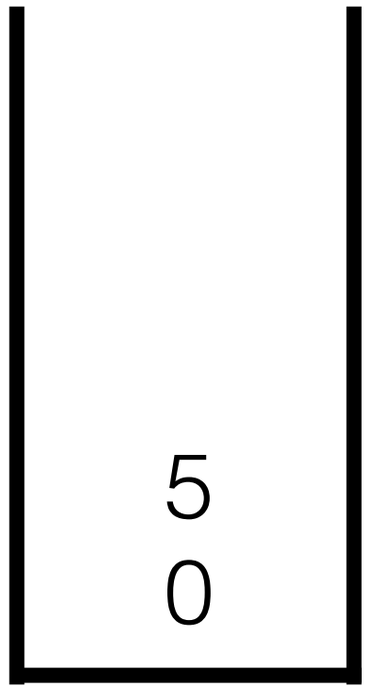




Added: 0 1 5

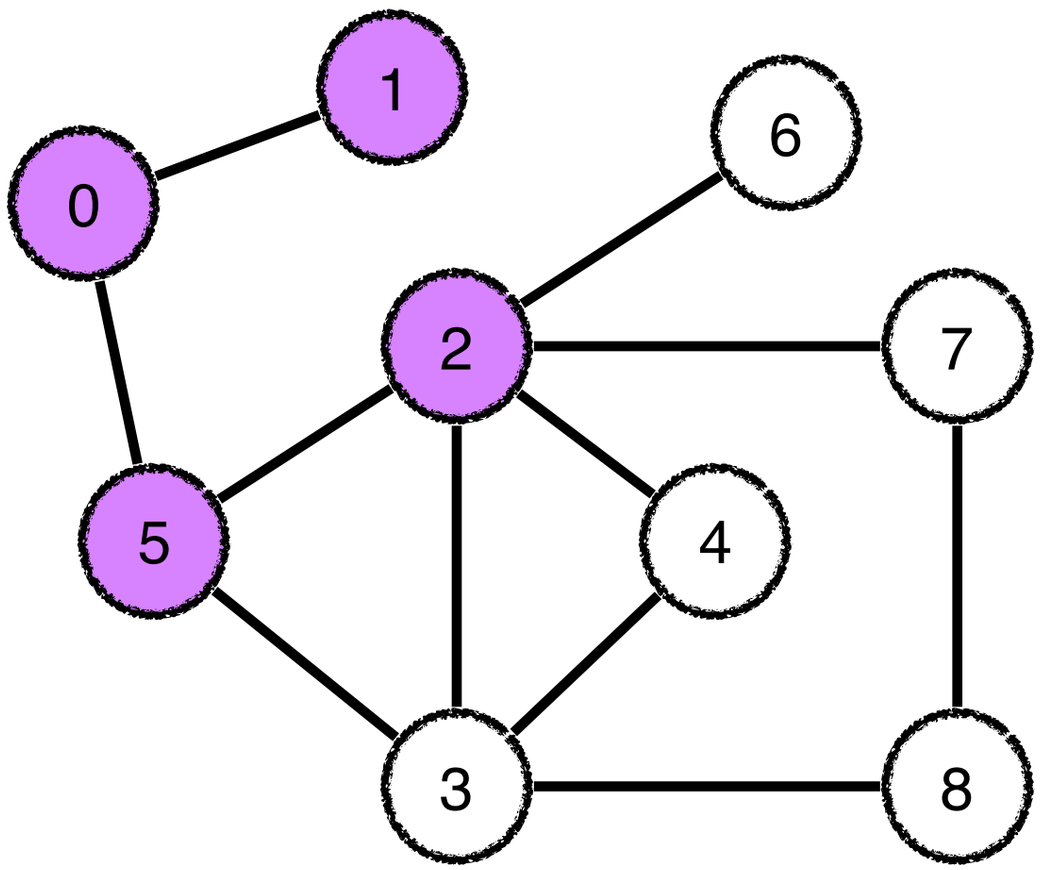
DFS

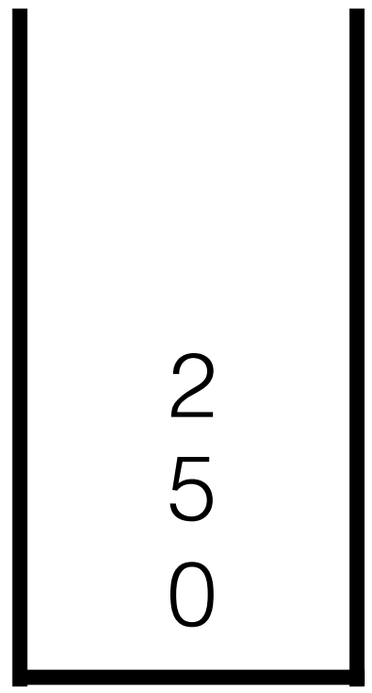




Added: 0 1 5

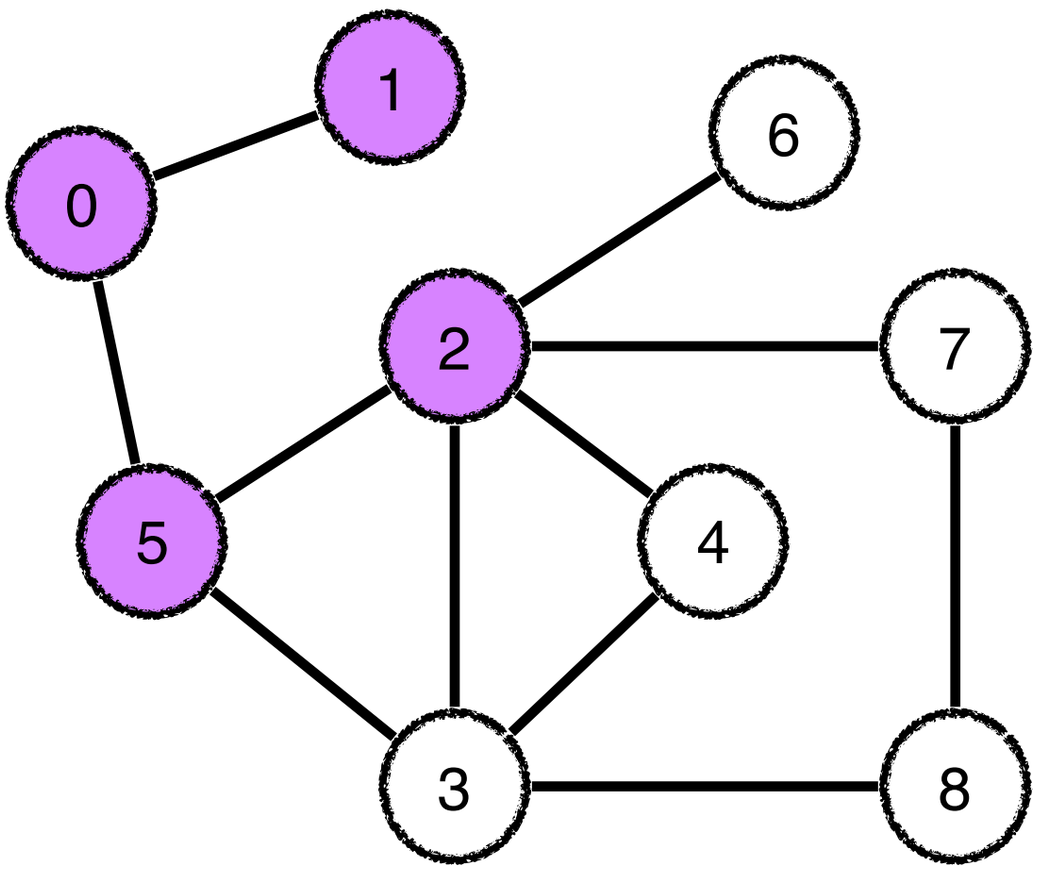
DFS

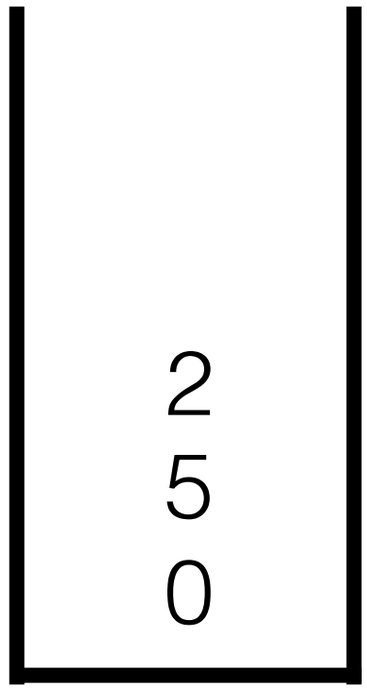




Added: 0 1 5

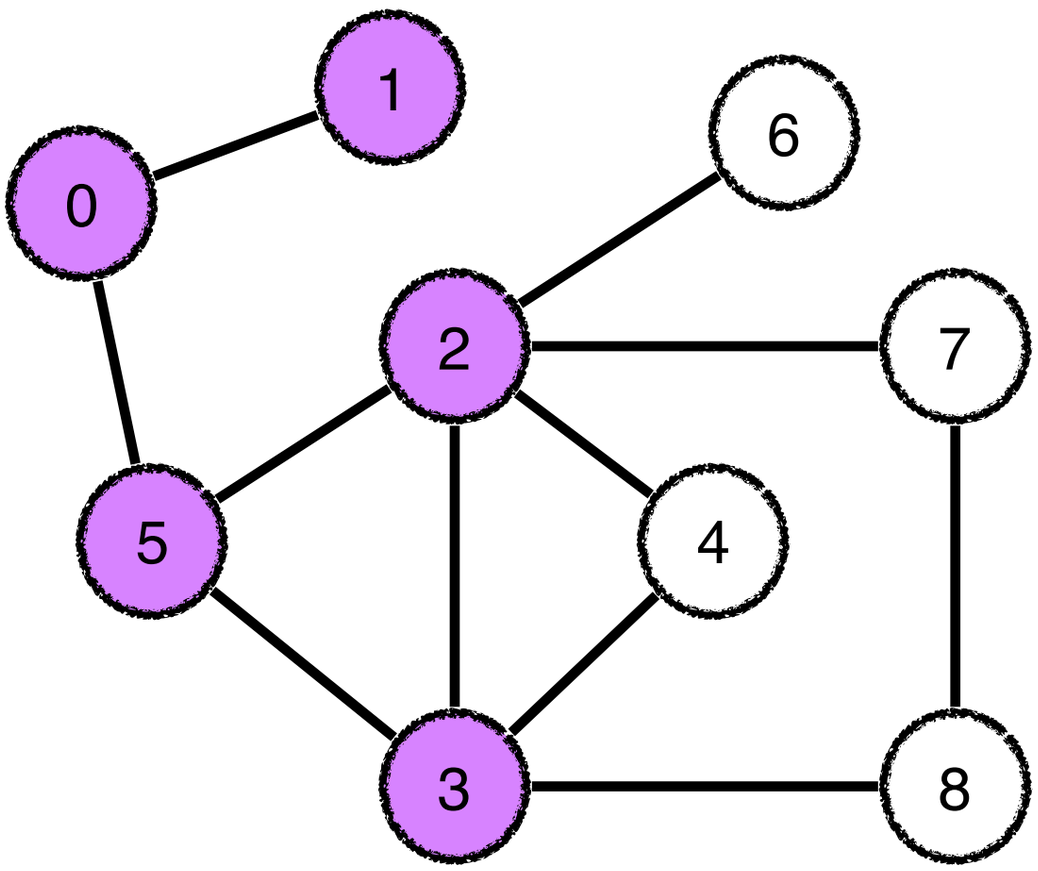
DFS

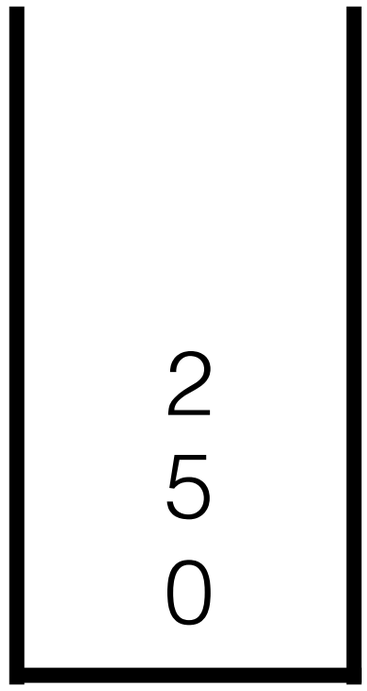




Added: 0 1 5 2

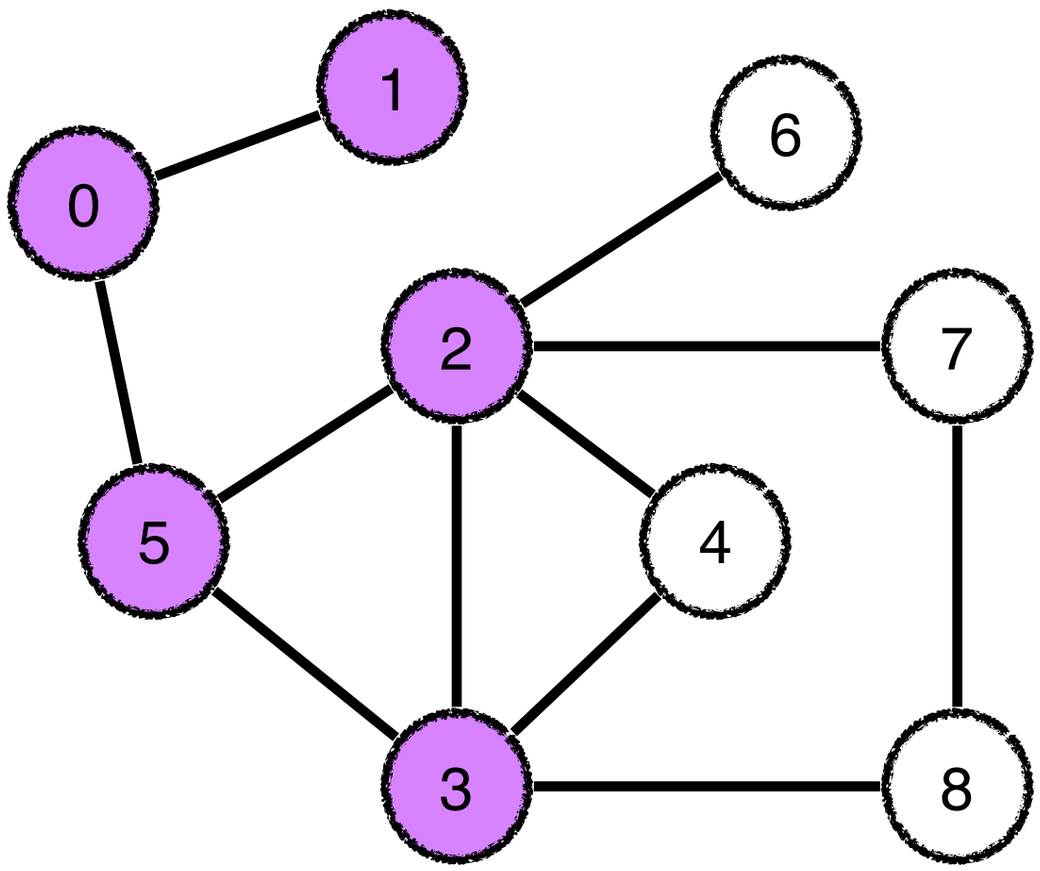
DFS

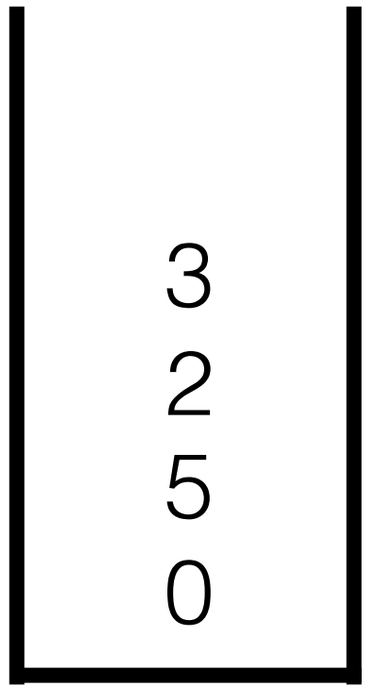




Added: 0 1 5 2

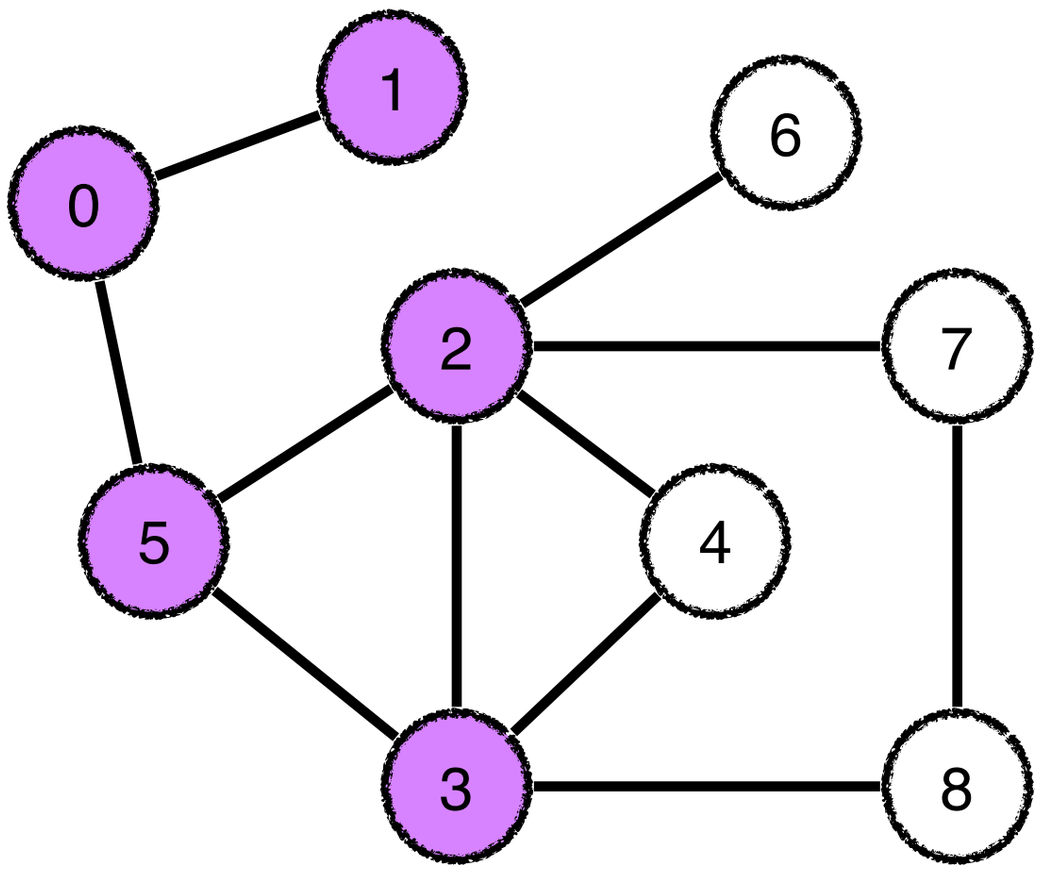
DFS

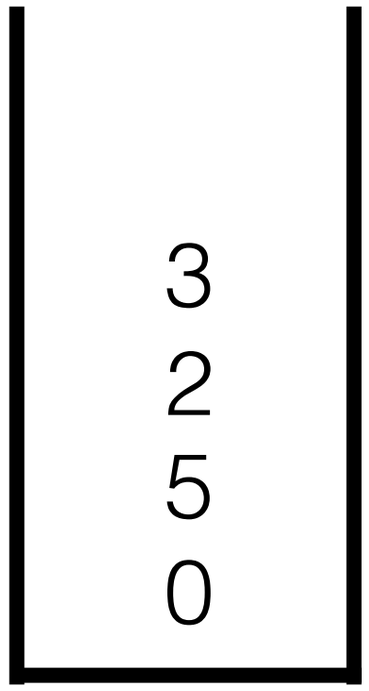




Added: 0 1 5 2

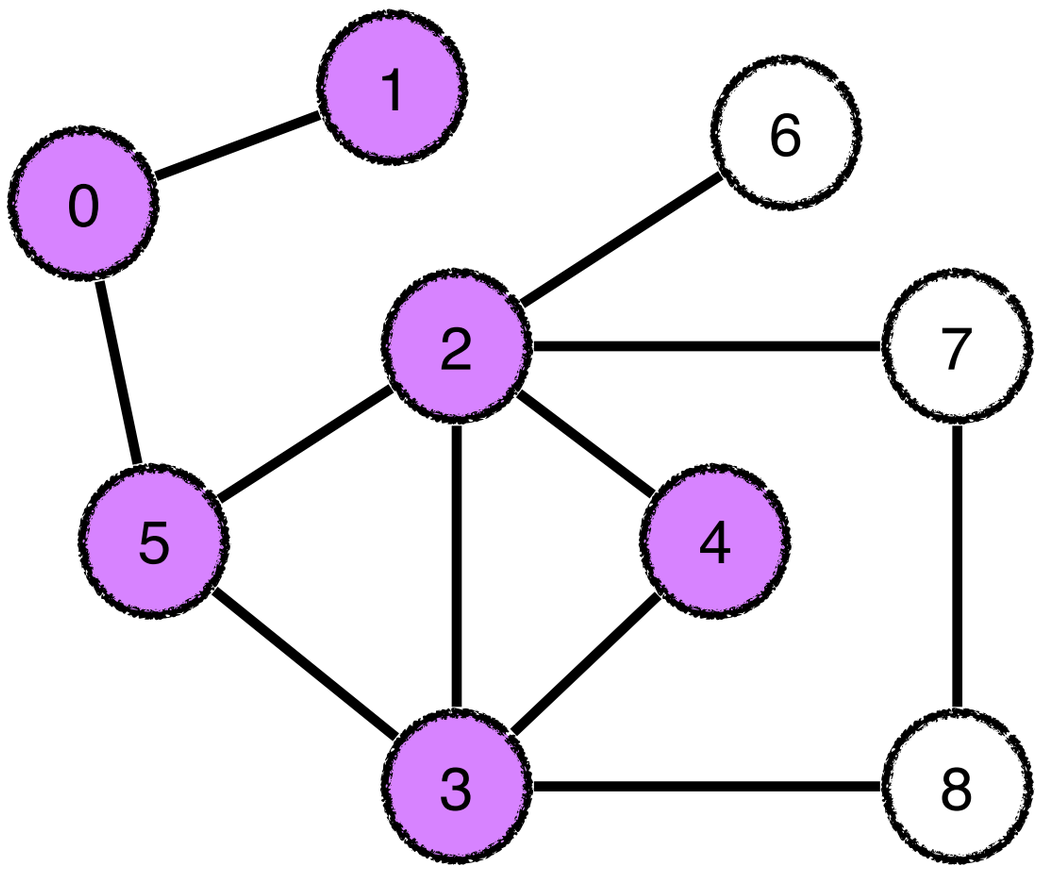
DFS

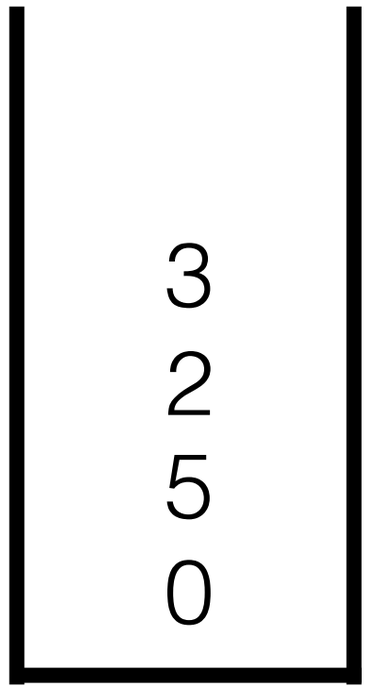




Added: 0 1 5 2 3

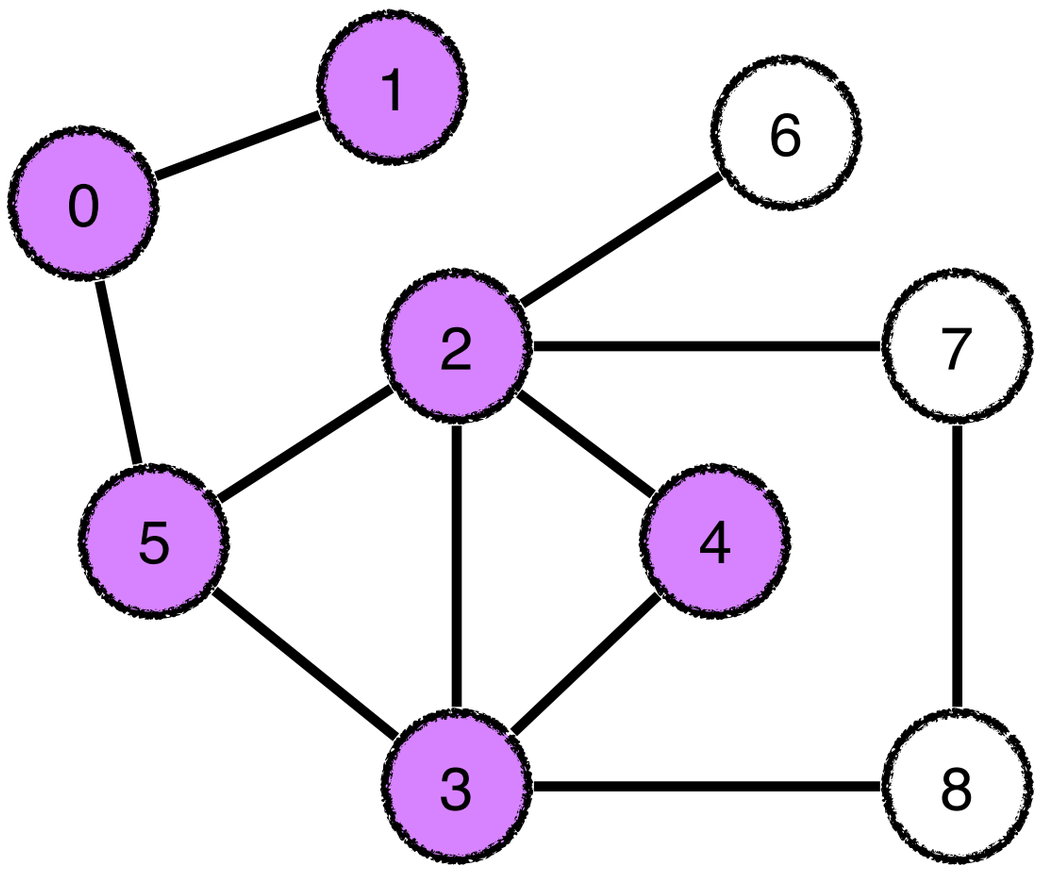
DFS

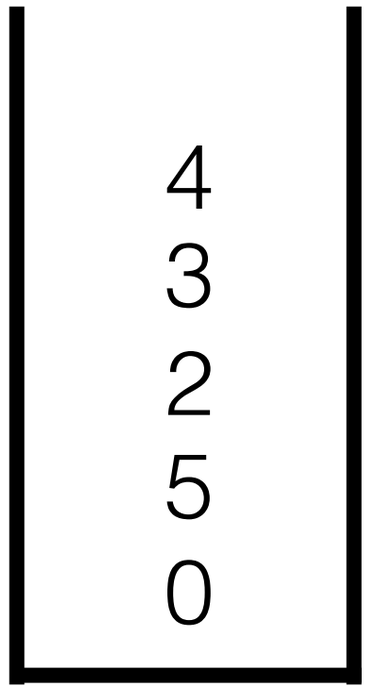




Added: 0 1 5 2 3

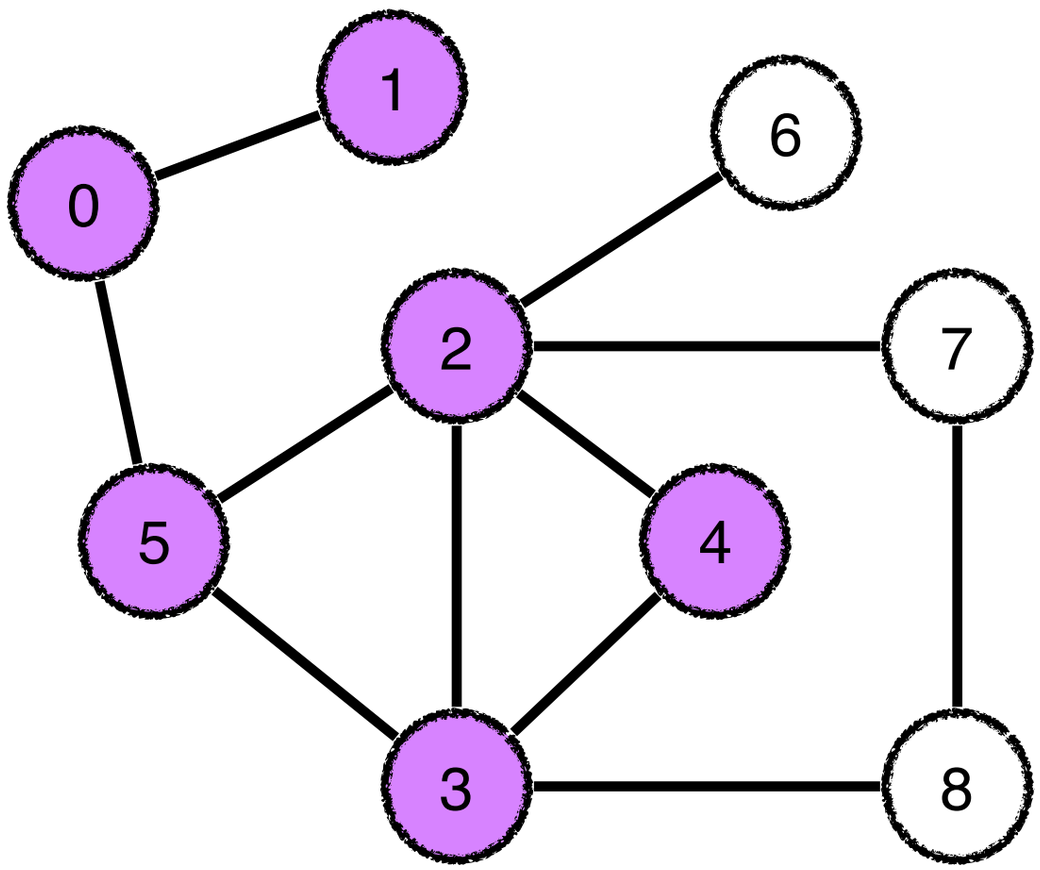
DFS

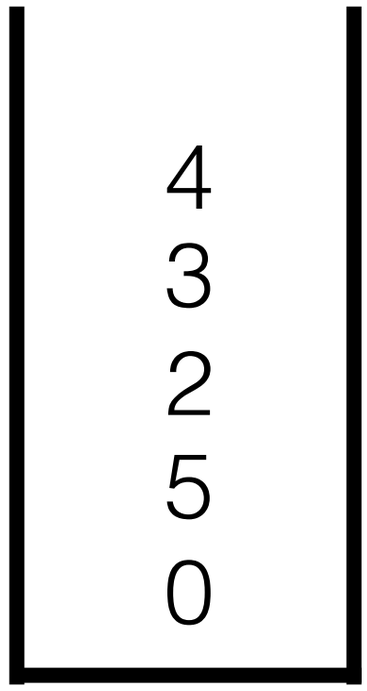




Added: 0 1 5 2 3

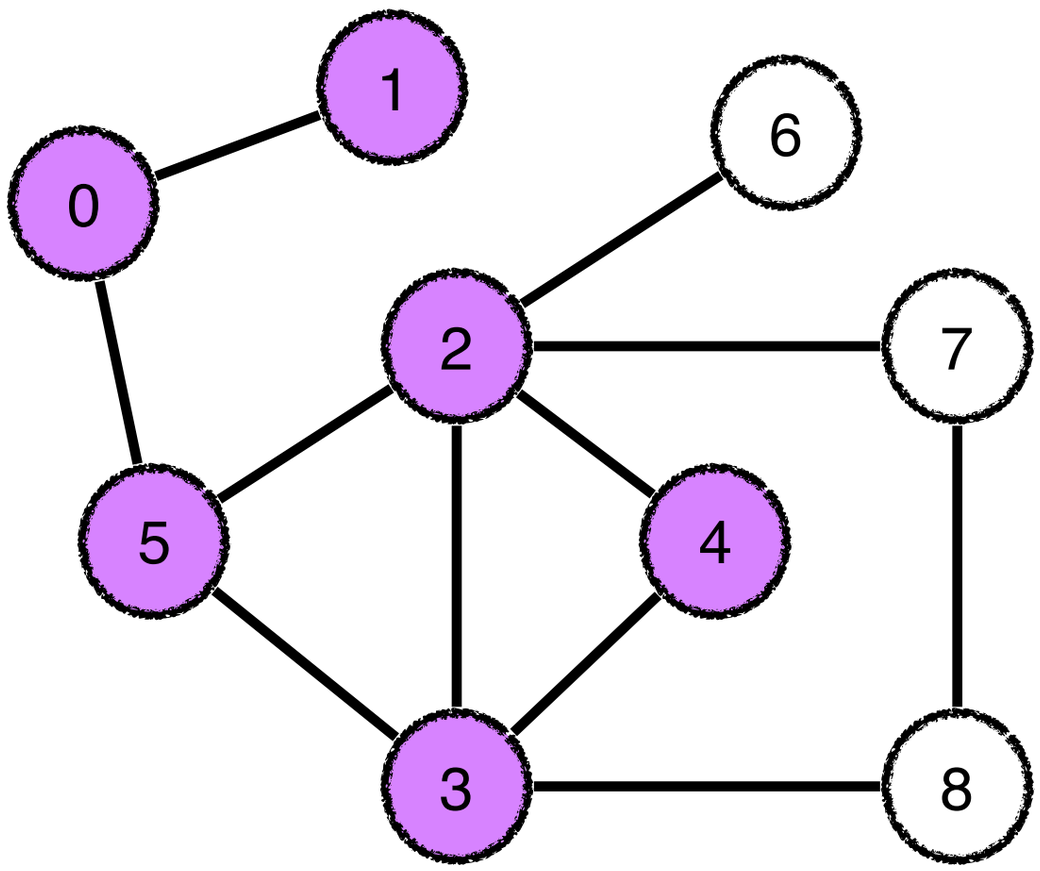
DFS

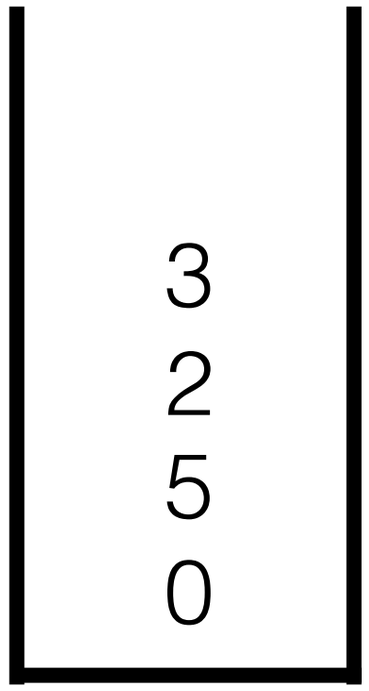




Added: 0 1 5 2 3 4

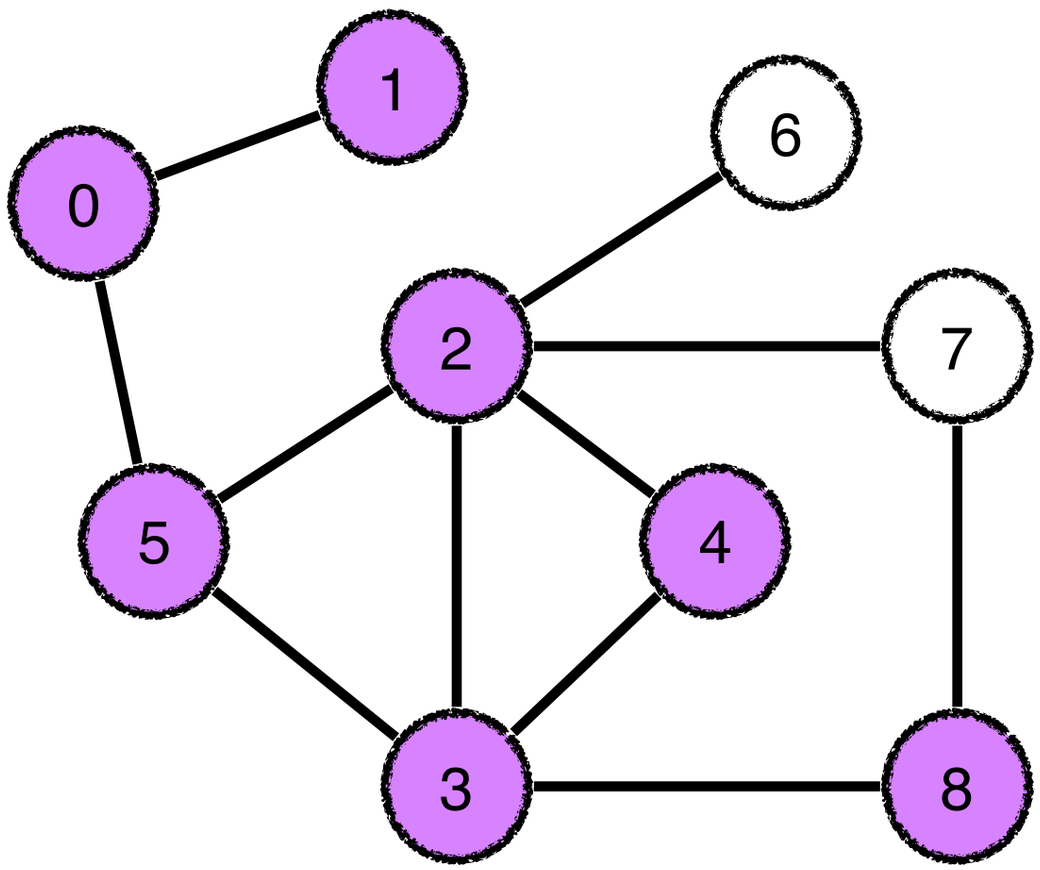
DFS

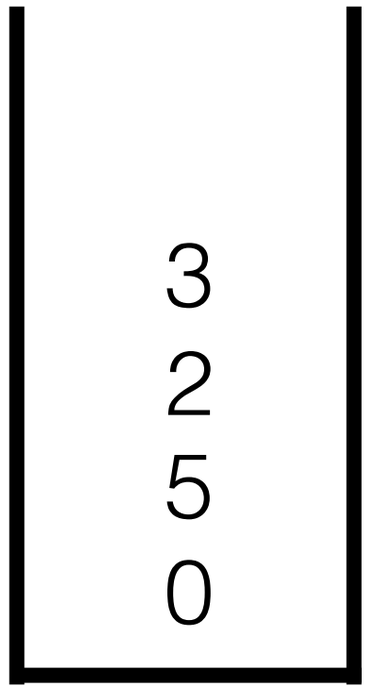




Added: 0 1 5 2 3 4

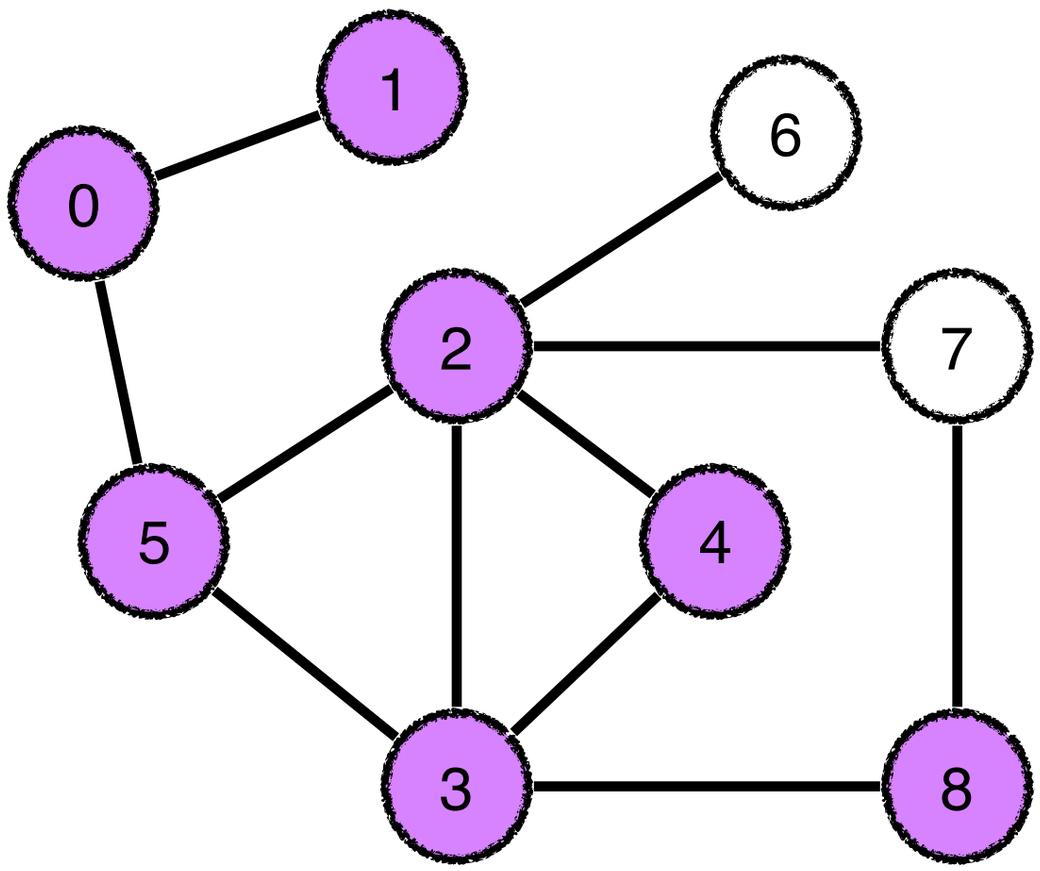
DFS

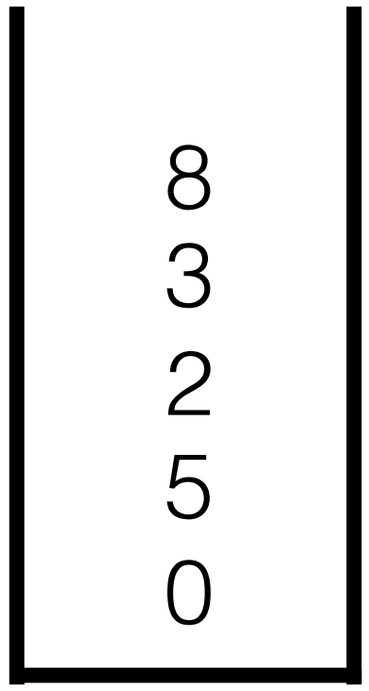




Added: 0 1 5 2 3 4

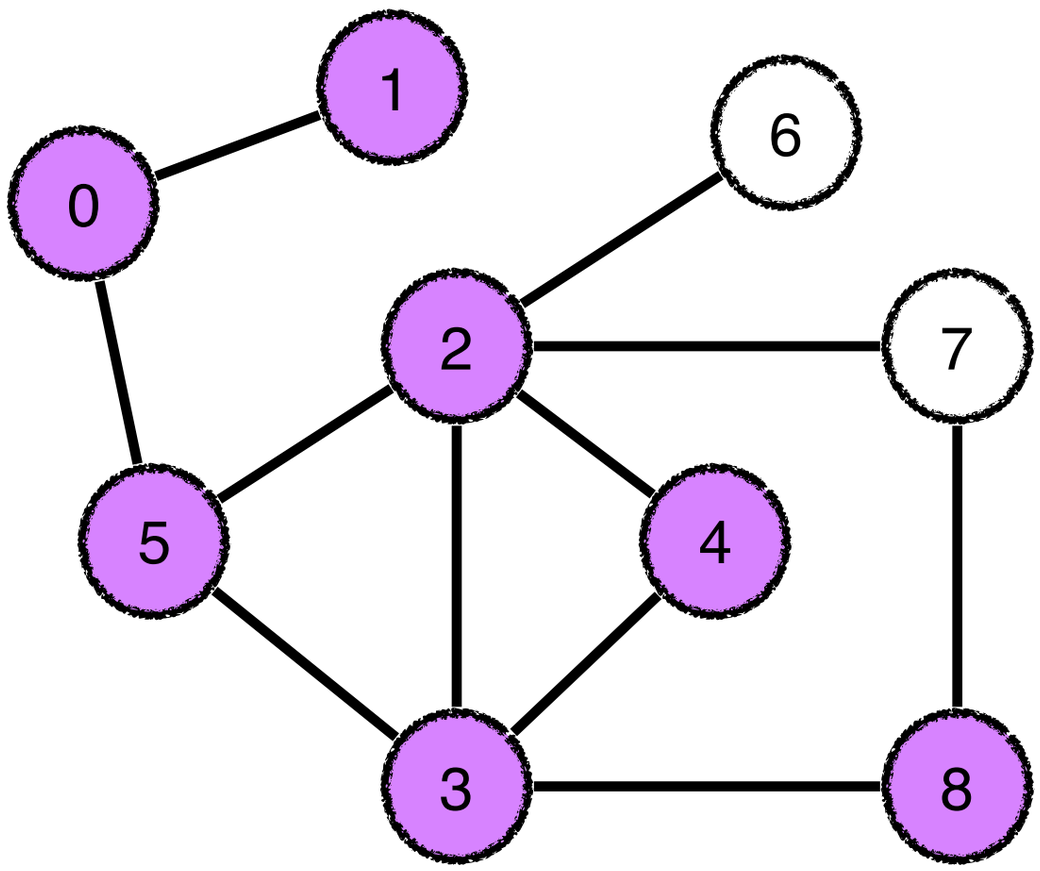
DFS

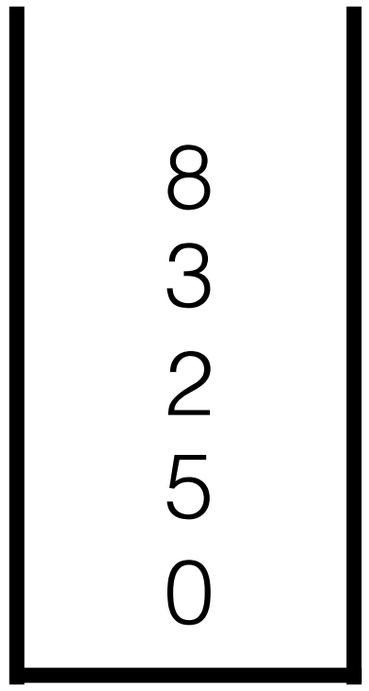




Added: 0 1 5 2 3 4

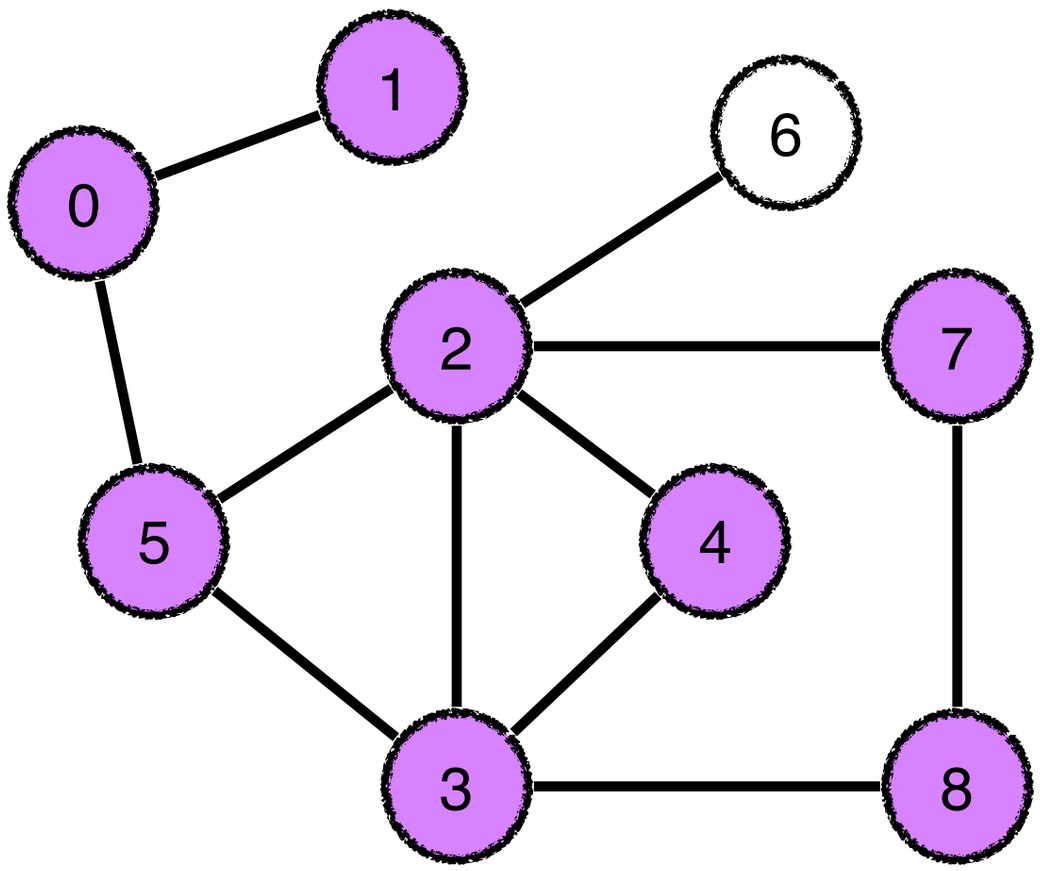
DFS

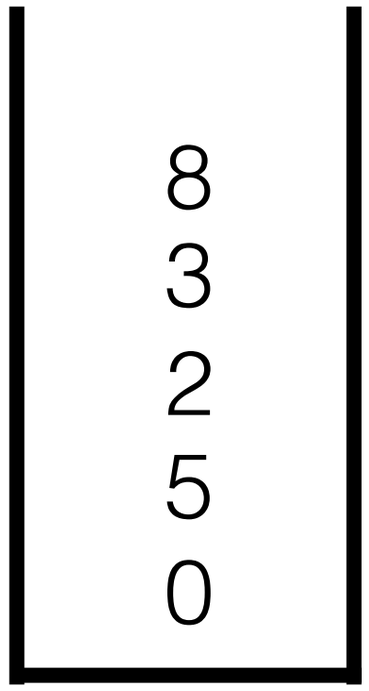




Added: 0 1 5 2 3 4 8

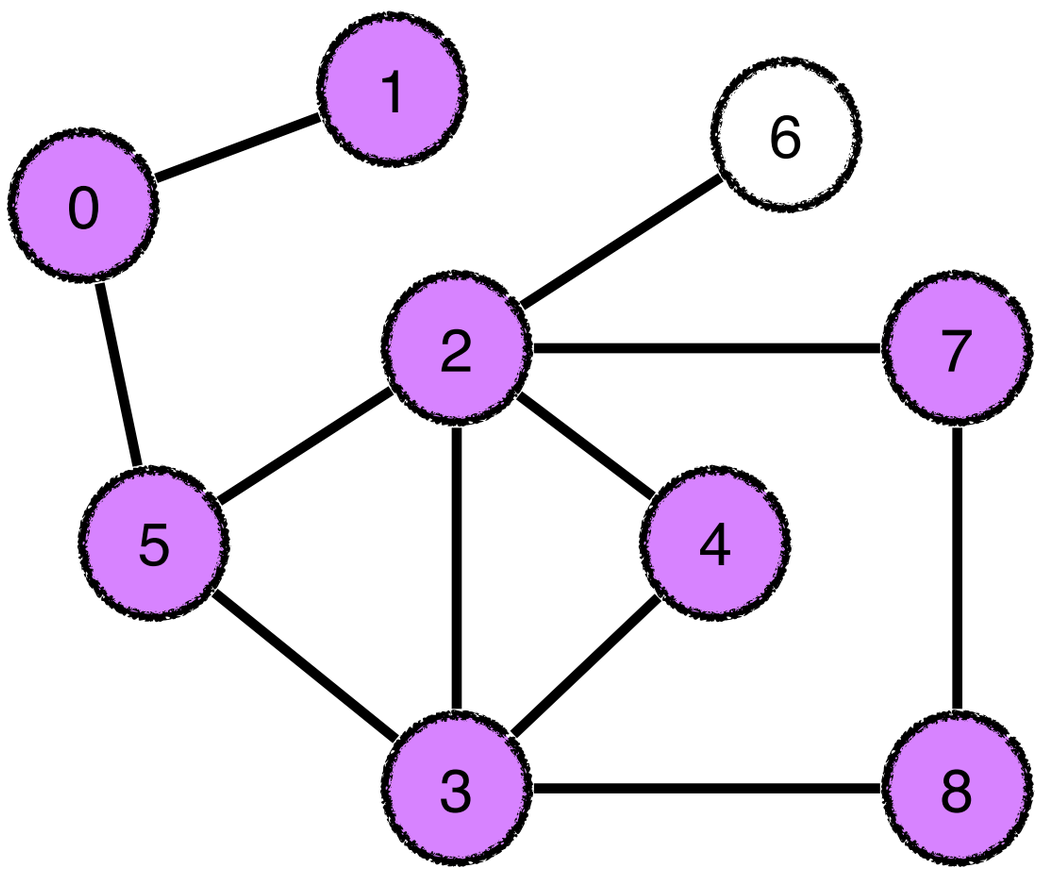
DFS

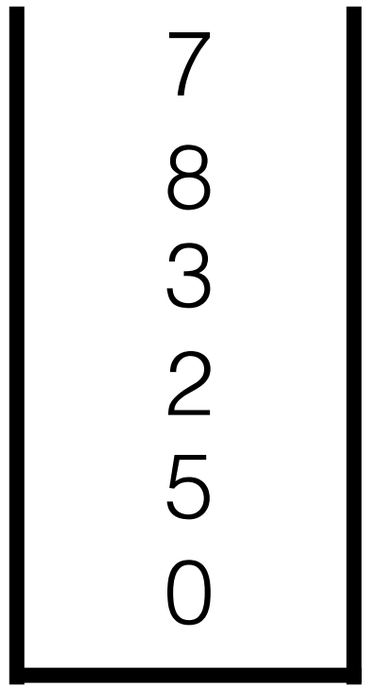




Added: 0 1 5 2 3 4 8

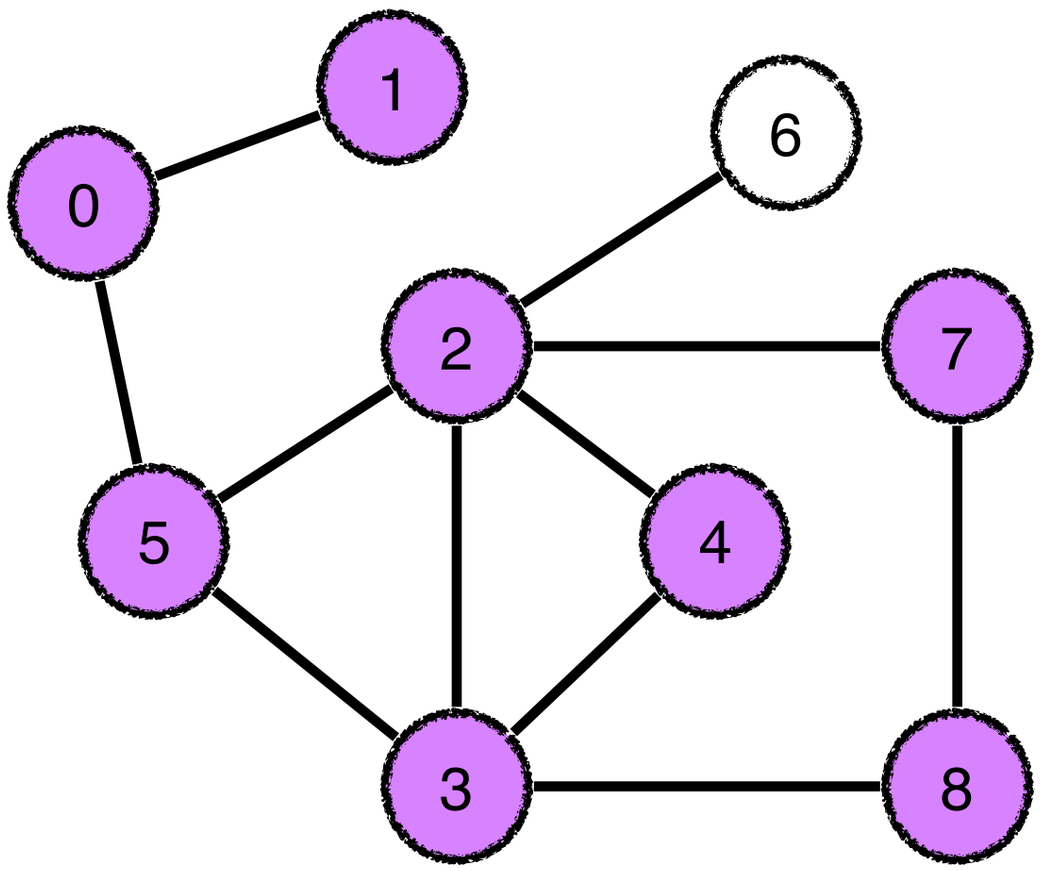
DFS

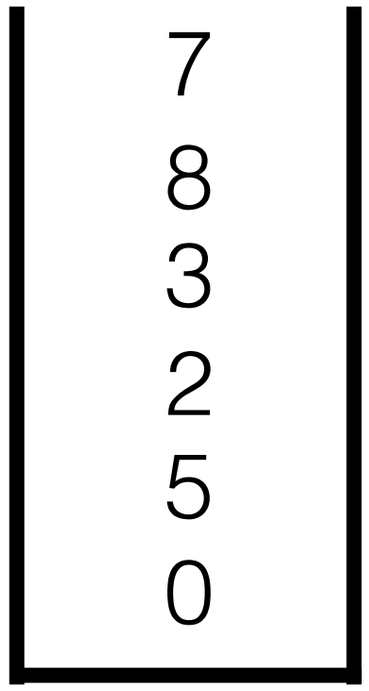




Added: 0 1 5 2 3 4 8

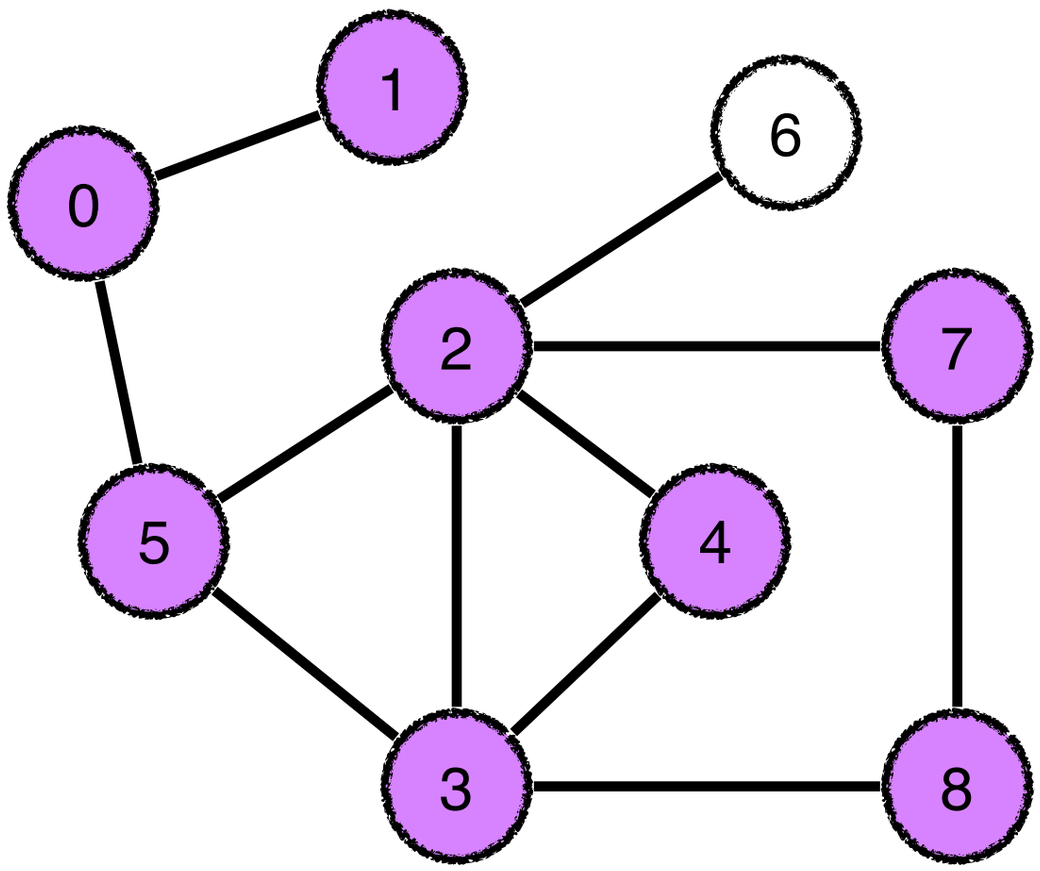
DFS

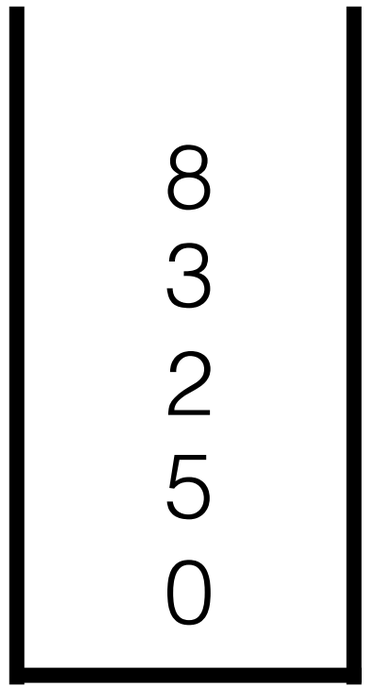




Added: 0 1 5 2 3 4 8 7

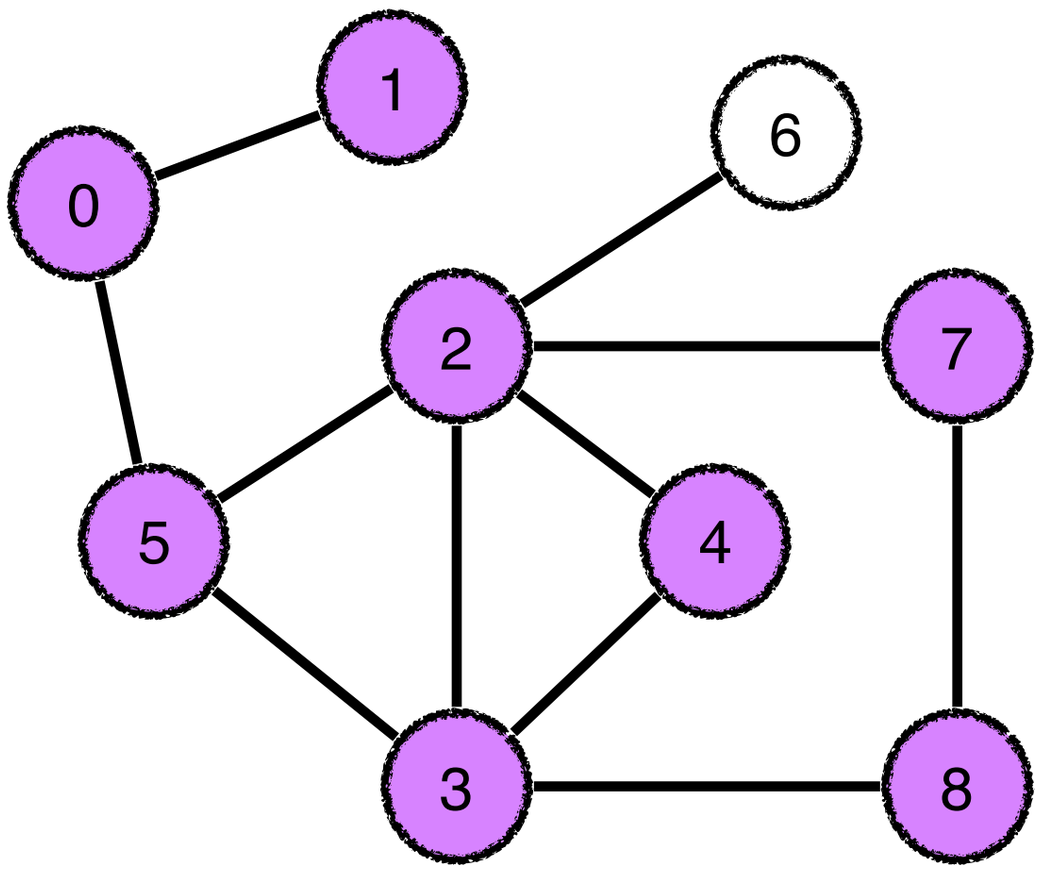
DFS

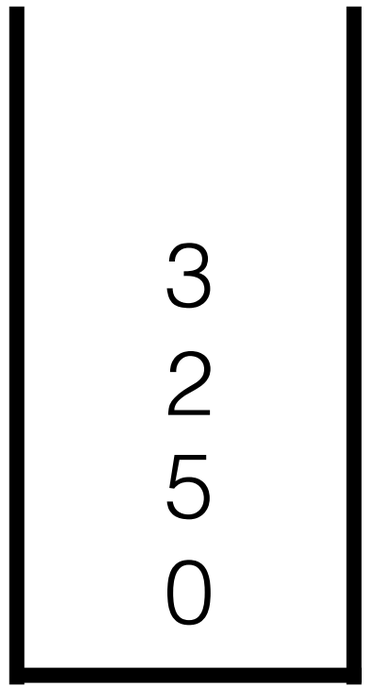




Added: 0 1 5 2 3 4 8 7

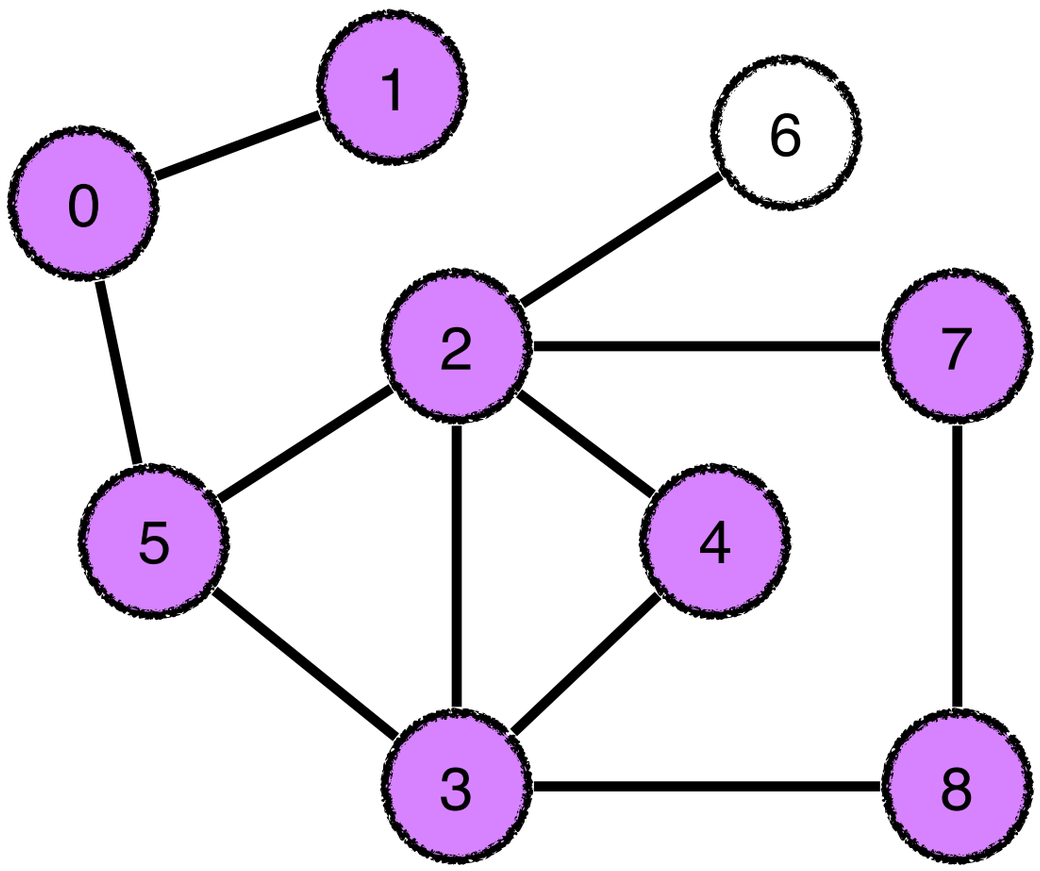
DFS

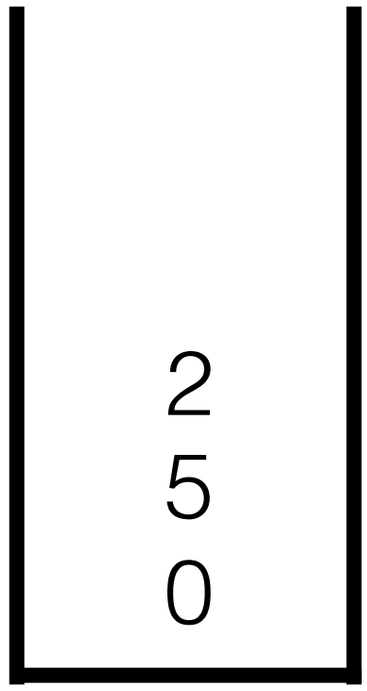




Added: 0 1 5 2 3 4 8 7

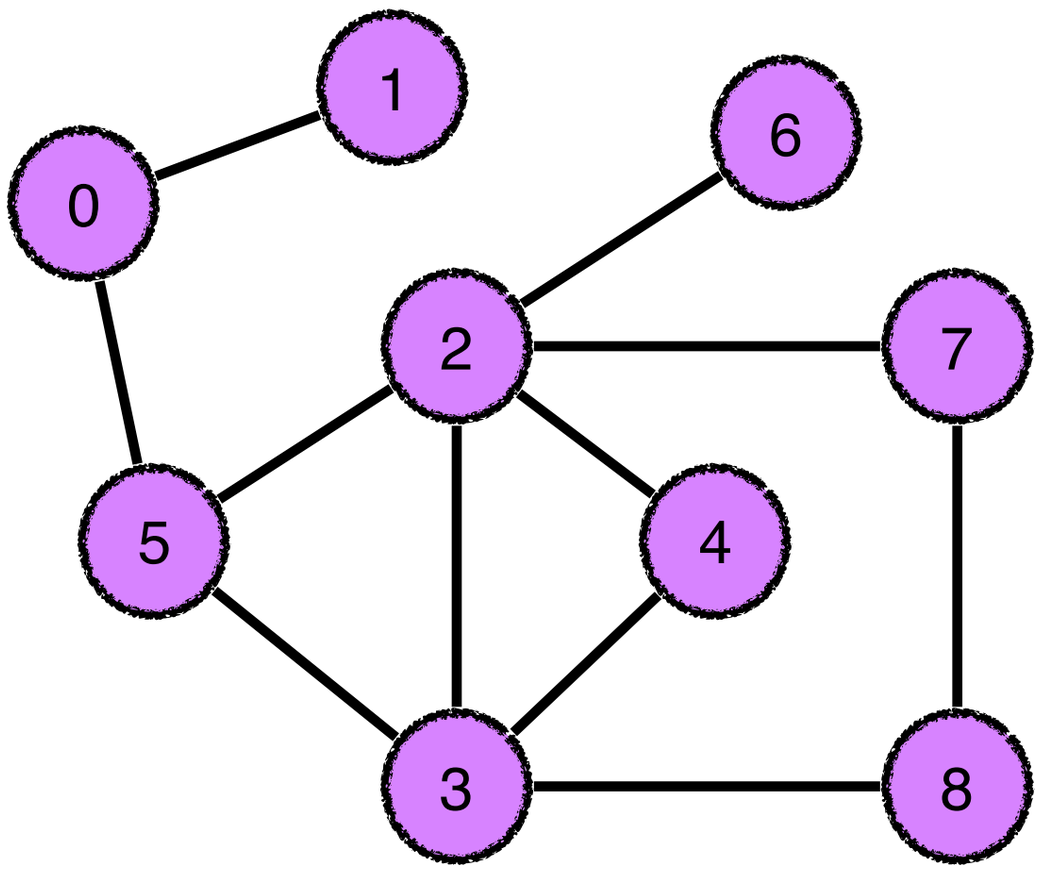
DFS

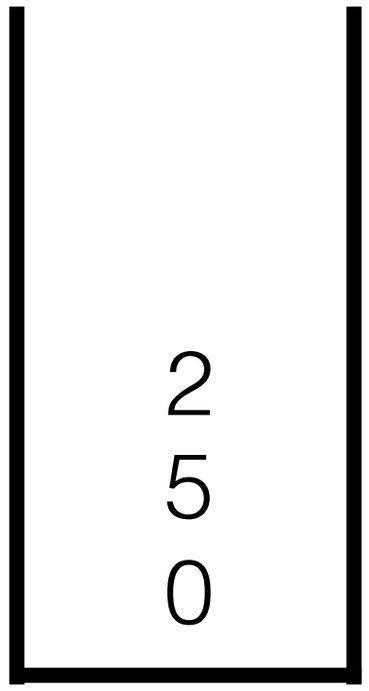




Added: 0 1 5 2 3 4 8 7

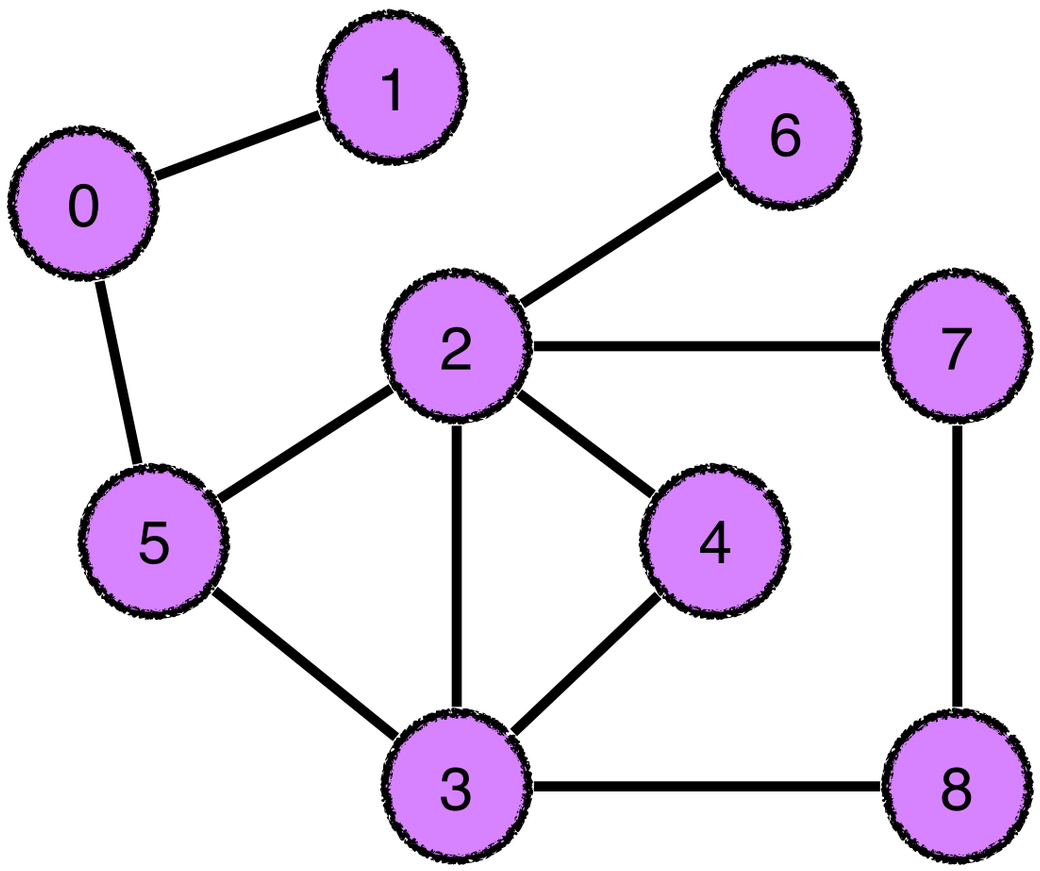
DFS

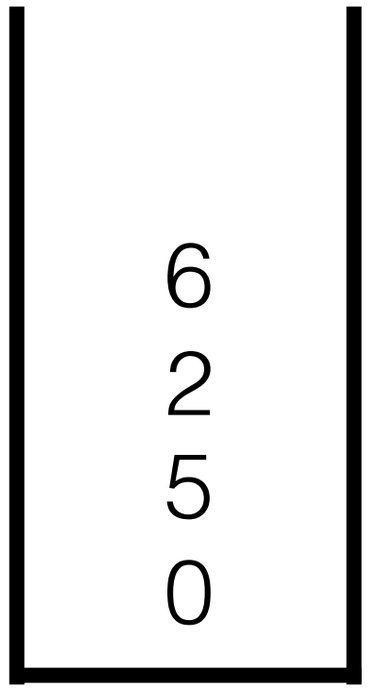




Added: 0 1 5 2 3 4 8 7

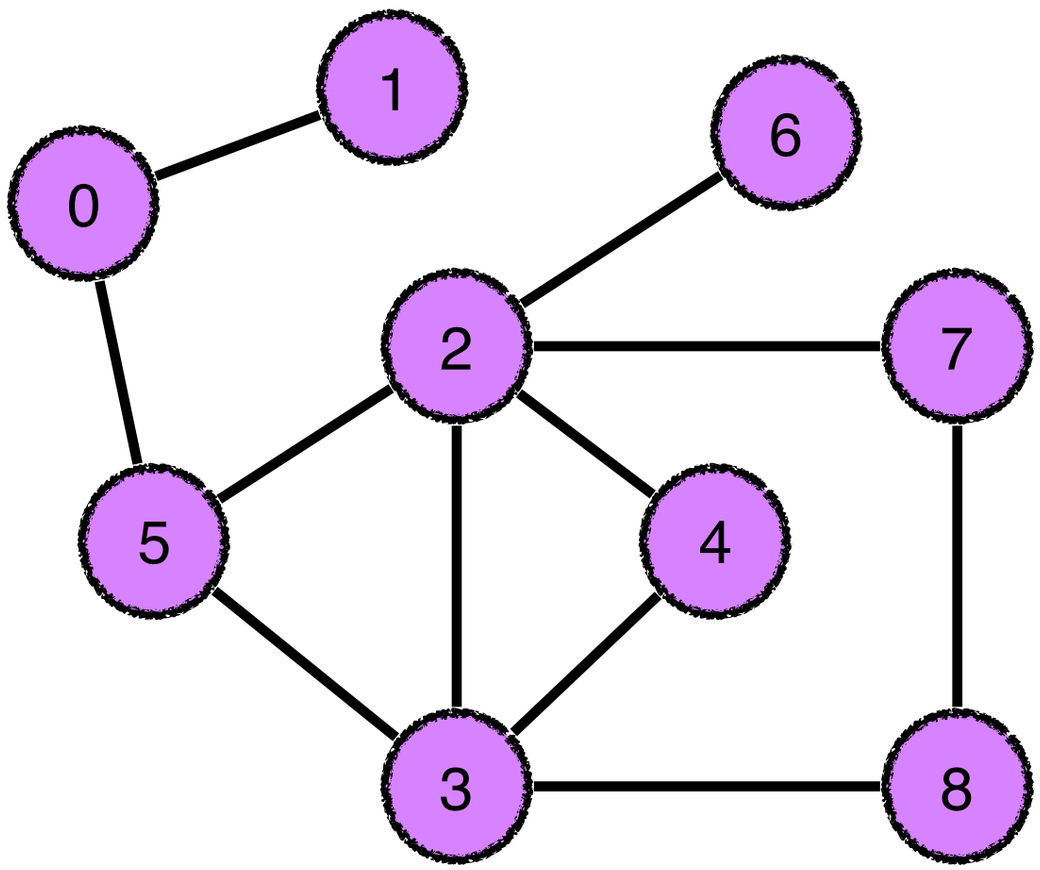
DFS

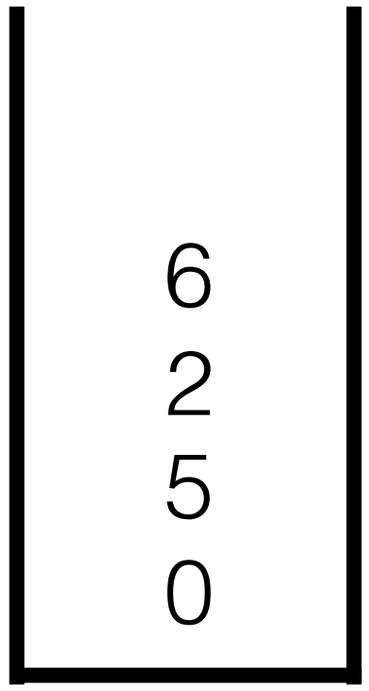




Added: 0 1 5 2 3 4 8 7

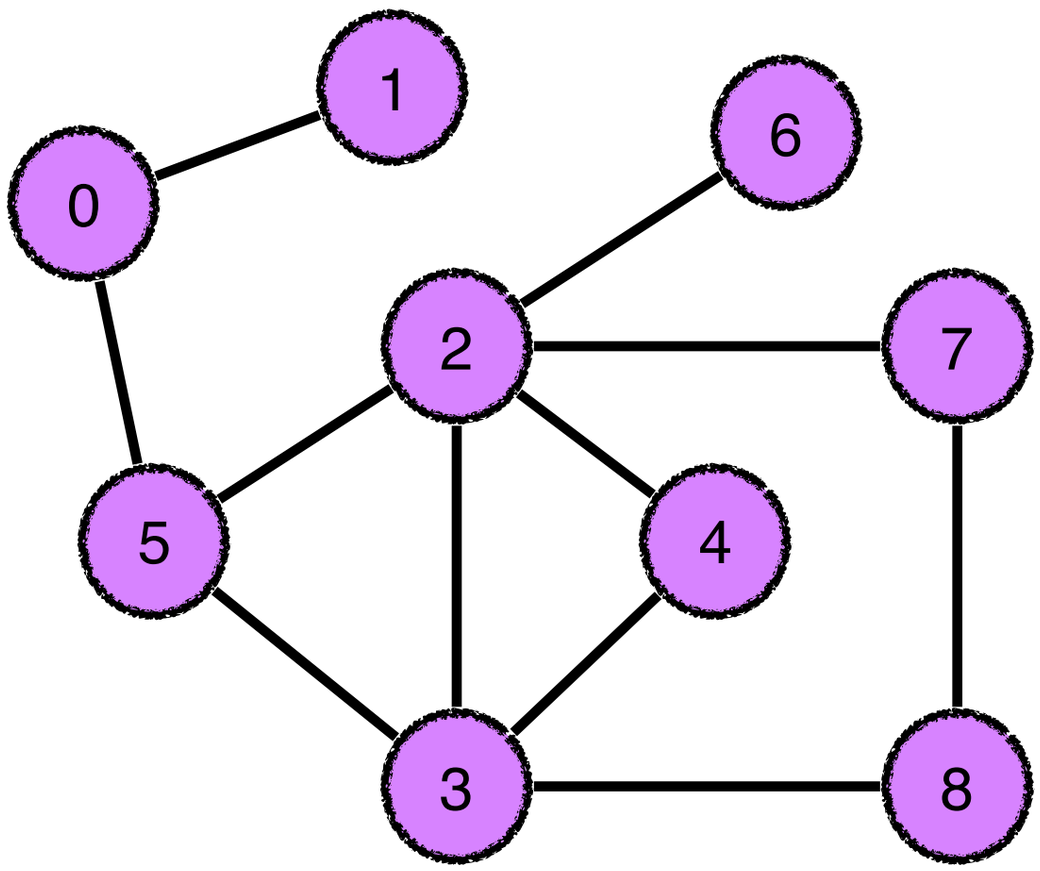
DFS

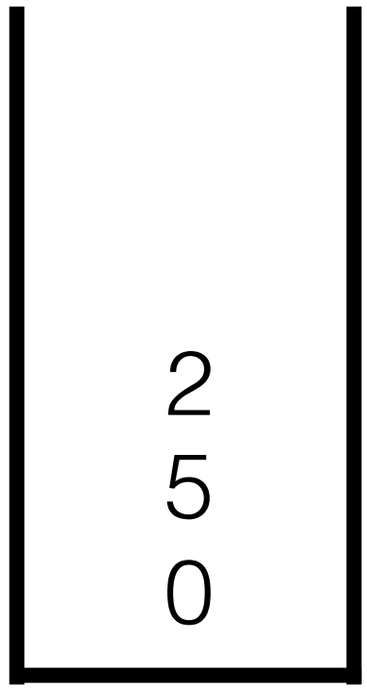




Added: 0 1 5 2 3 4 8 7 6

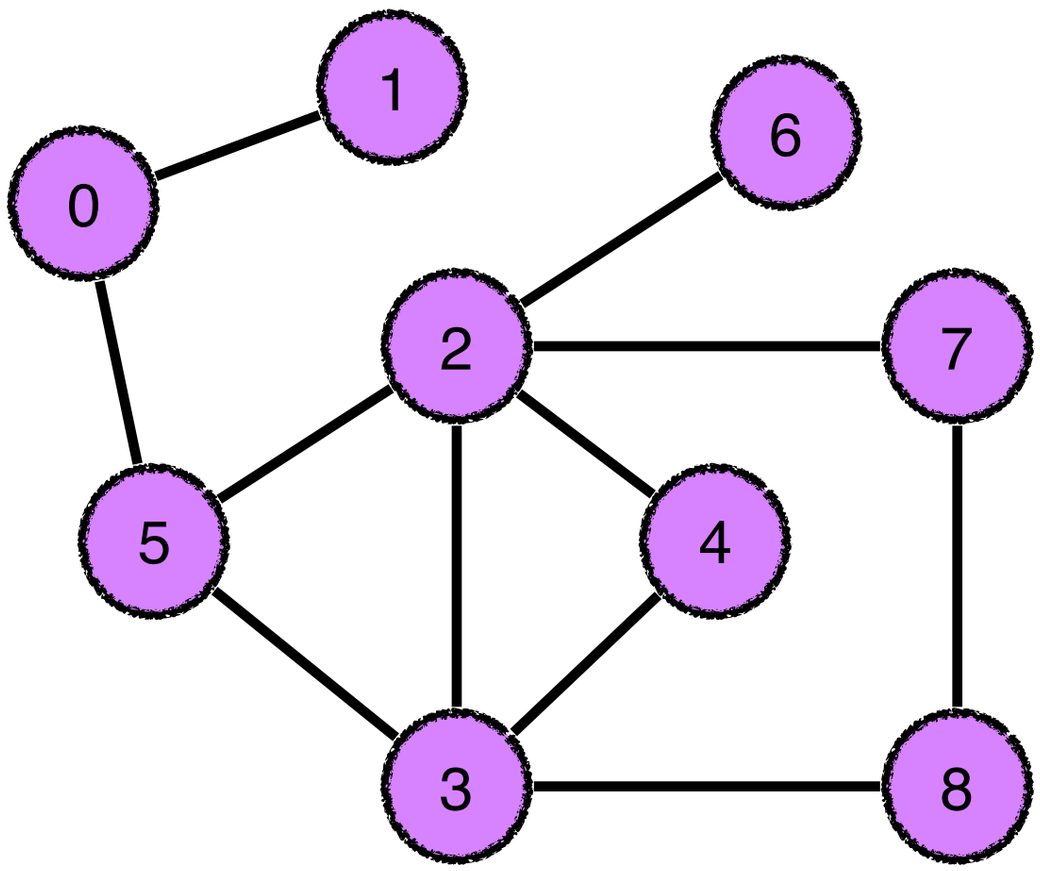
DFS

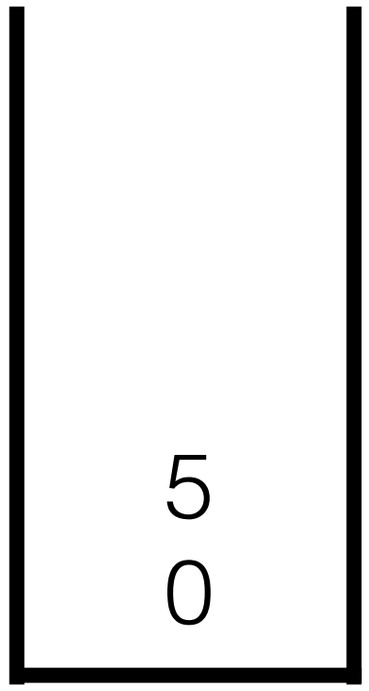




Added: 0 1 5 2 3 4 8 7 6

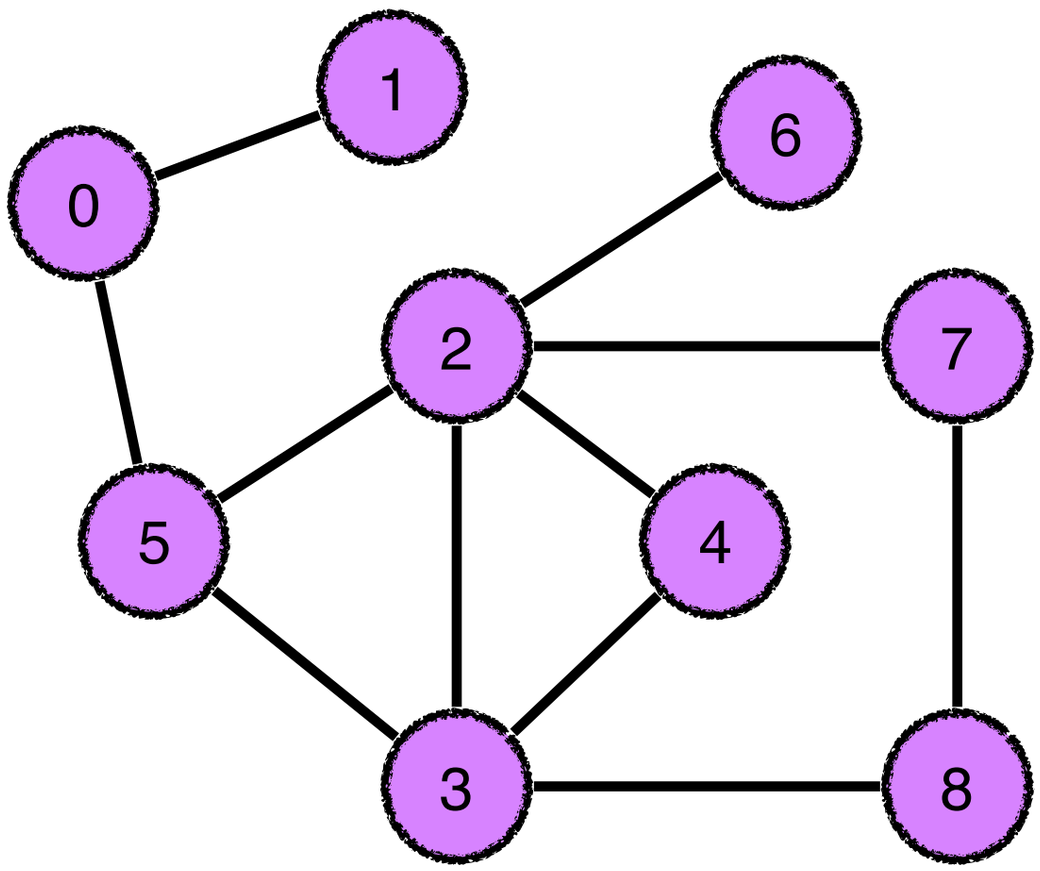
DFS

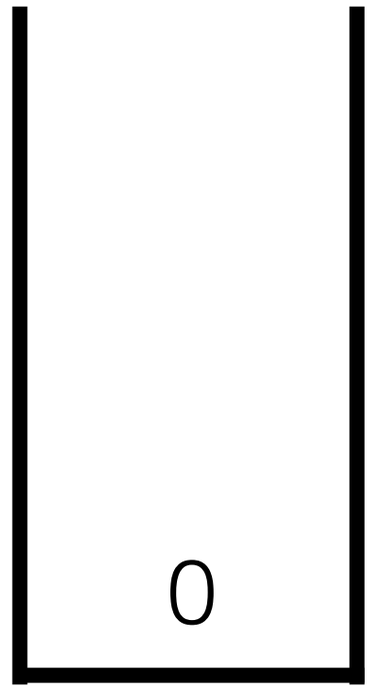




Added: 0 1 5 2 3 4 8 7 6

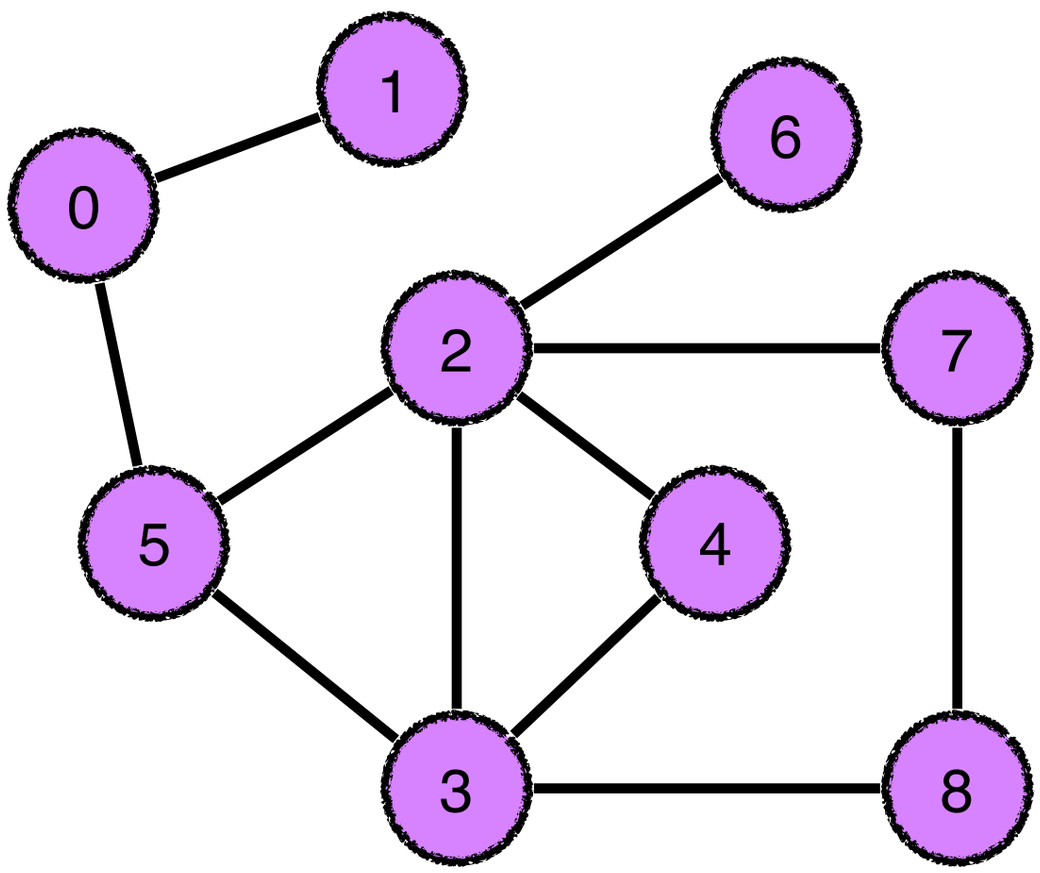
DFS

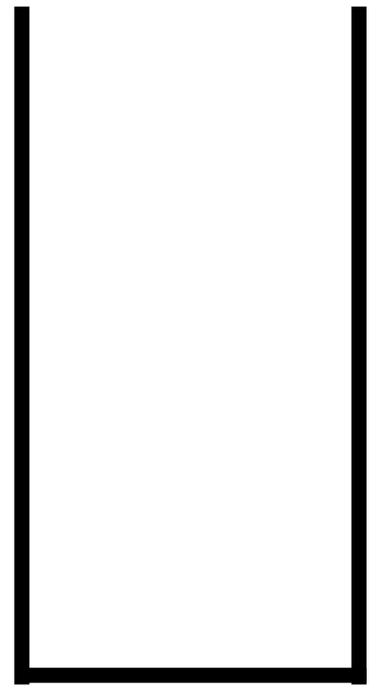




Added: 0 1 5 2 3 4 8 7 6

DFS





Added: 0 1 5 2 3 4 8 7 6

DFS

```
void dfs(vector<int> nodes[], int v) {
    bool *visited = new bool[nodes->size()];
    for (int i = 0; i < nodes->size(); i++) {
        visited[i] = false;
    }
    dfsUtil(nodes, v, visited);
}
```

mark vertices as not visited

```
    void dfsUtil(vector<int> nodes[], int v, bool visited[]) {
        visited[v] = true;
        cout << v << " ";

        list<int>::iterator i;
        for (auto u : nodes[v])
            if (!visited[u])
                dfsUtil(nodes, u, visited);
    }
```

visited

go down adjacent nodes

https://codeforces.com/problemset/problem/500/A

New Year is coming in Line World! In this world, there are n cells numbered by integers from 1 to n, as a 1 × n board. People live in cells. However, it was hard to move between distinct cells, because of the difficulty of escaping the cell. People wanted to meet people who live in other cells.

So, user tncks0121 has made a transportation system to move between these cells, to celebrate the New Year. First, he thought of n - 1 positive integers a1, a2, ..., an - 1. For every integer i where $1 \le i \le n - 1$ the condition $1 \le a_i \le n - i$ holds. Next, he made n - 1 portals, numbered by integers from 1 to n - 1. The i-th ($1 \le i \le n - 1$) portal connects cell i and cell (i + ai), and one can travel from cell i to cell (i + ai) using the i-th portal. Unfortunately, one cannot use the portal backwards, which means one cannot move from cell (i + ai) to cell i using the i-th portal. It is easy to see that because of condition $1 \le a_i \le n - i$ one can't leave the Line World using portals.

Currently, I am standing at cell 1, and I want to go to cell t. However, I don't know whether it is possible to go there. Please determine whether I can go to cell t by only using the constructed transportation system.

https://codeforces.com/problemset/problem/500/A

Input
The first line contains two space-separated integers n ($3 \le n \le 3 \times 104$) and t ($2 \le t \le n$) — the number of cells, and the index of the cell which I want to go to.

The second line contains n - 1 space-separated integers a1, a2, ..., an - 1 ($1 \le ai \le n - i$). It is guaranteed, that using the given transportation system, one cannot leave the Line World.

Output
If I can go to cell t using the transportation system, print "YES". Otherwise, print "NO".

https://codeforces.com/problemset/problem/500/A

Floyd-Warshall

Objective

Shortest path with negative arcs, but no negative cycles

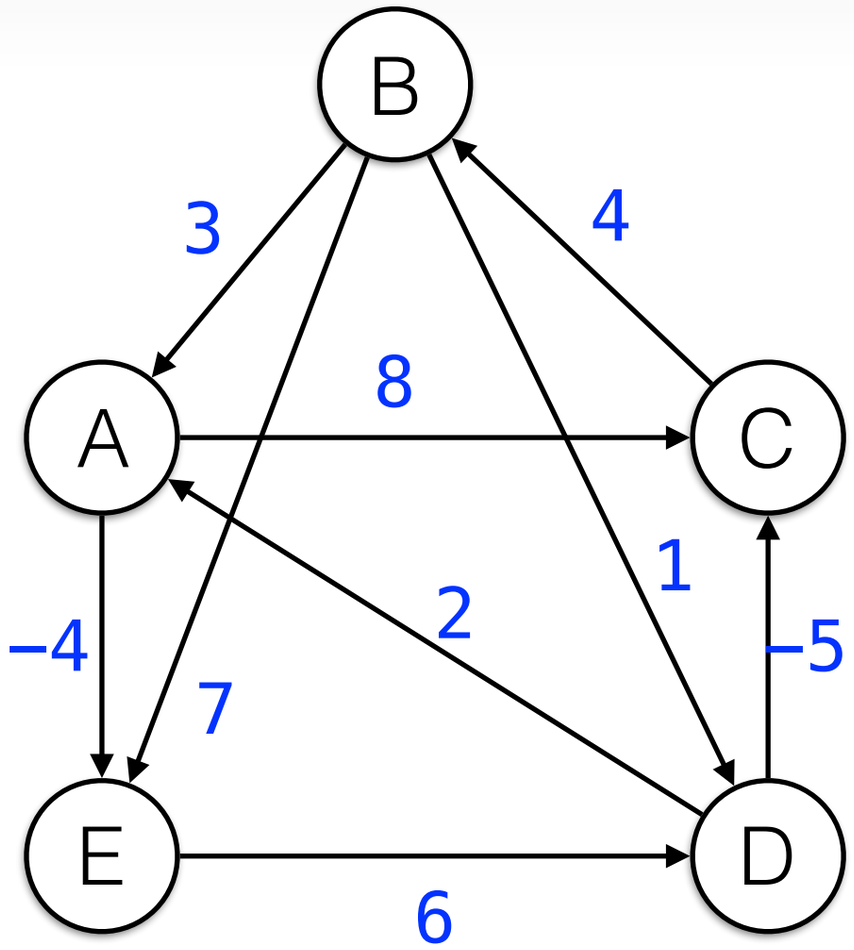
How to

Build the paths (in a matrix) for all node pairs, for every node in the graph

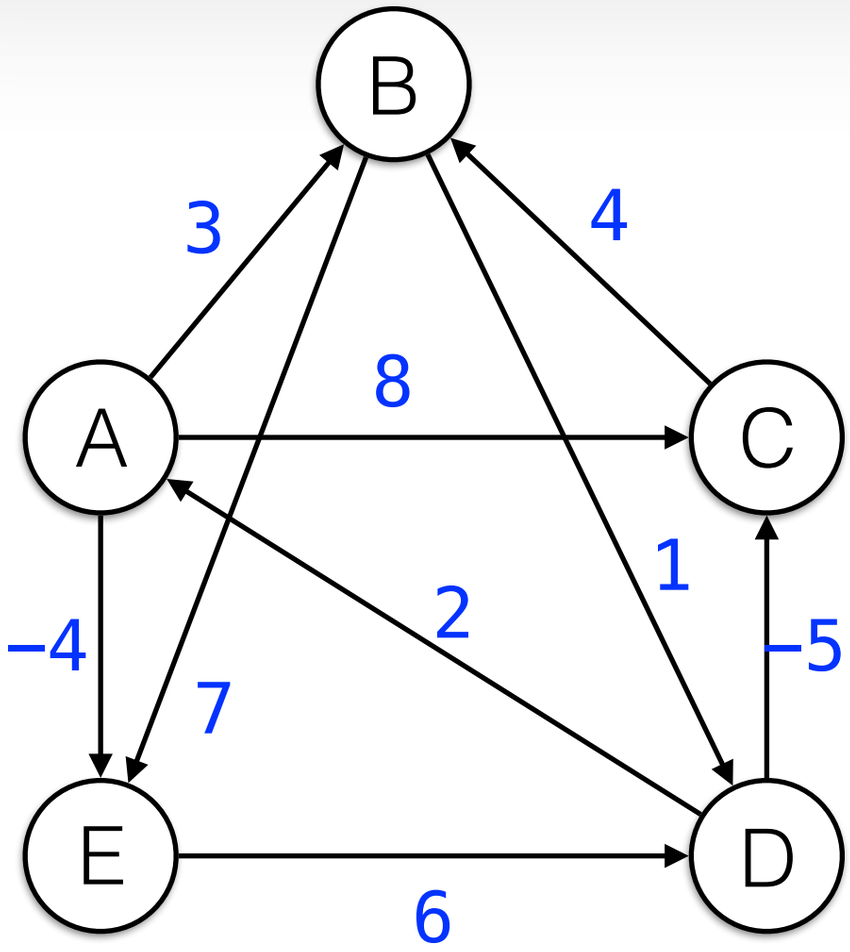
Floyd-Warshall

```
public void floydWarshall(V s) {
  int N = G.vertices().size();
  Double d[N][N]={Double.POSITIVE_INFINITY};
  for(V v : g.vertices()) d[v][v] = 0;
  for(E e : G.edges()) d[e.source()][e.dest()] = e.weight();
  for(k=0; k<N; k++) {
   for(i=0; i<N; i++){
    for(j=0; j<N; j++){
       if(d[i][j] > d[i][k] + d[k][j])
        d[i][j] = d[i][k] + d[k][j]
    }
   }
  }
}
```

Floyd-Warshall

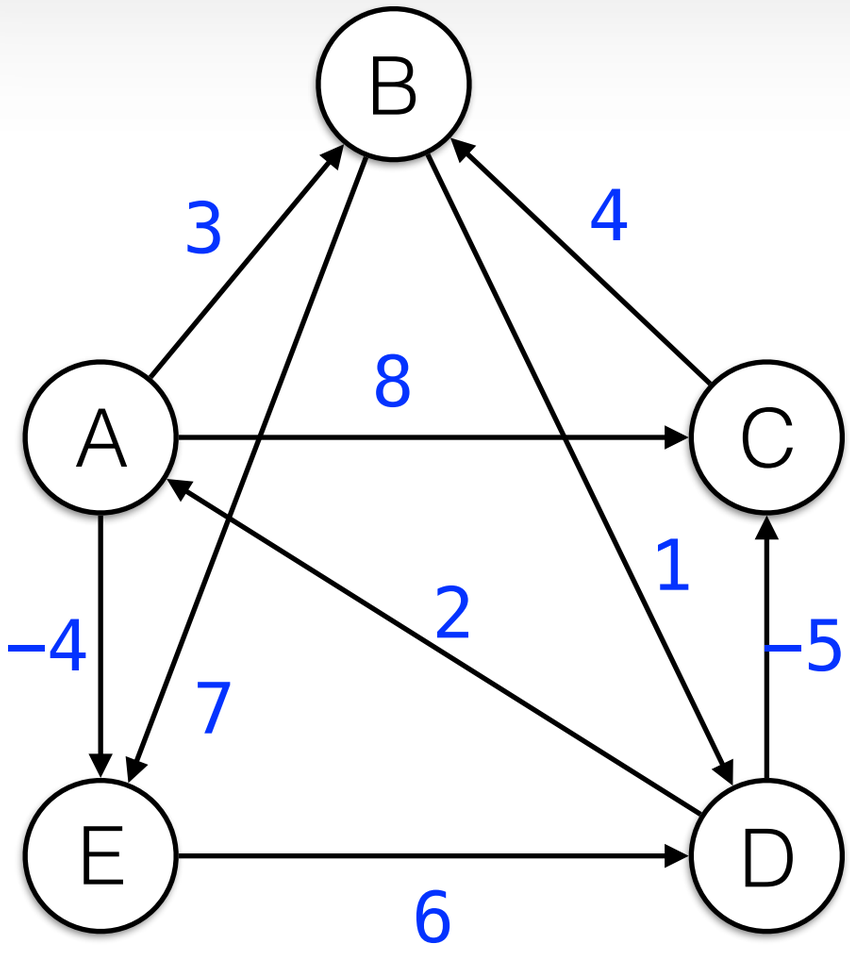


Floyd-Warshall



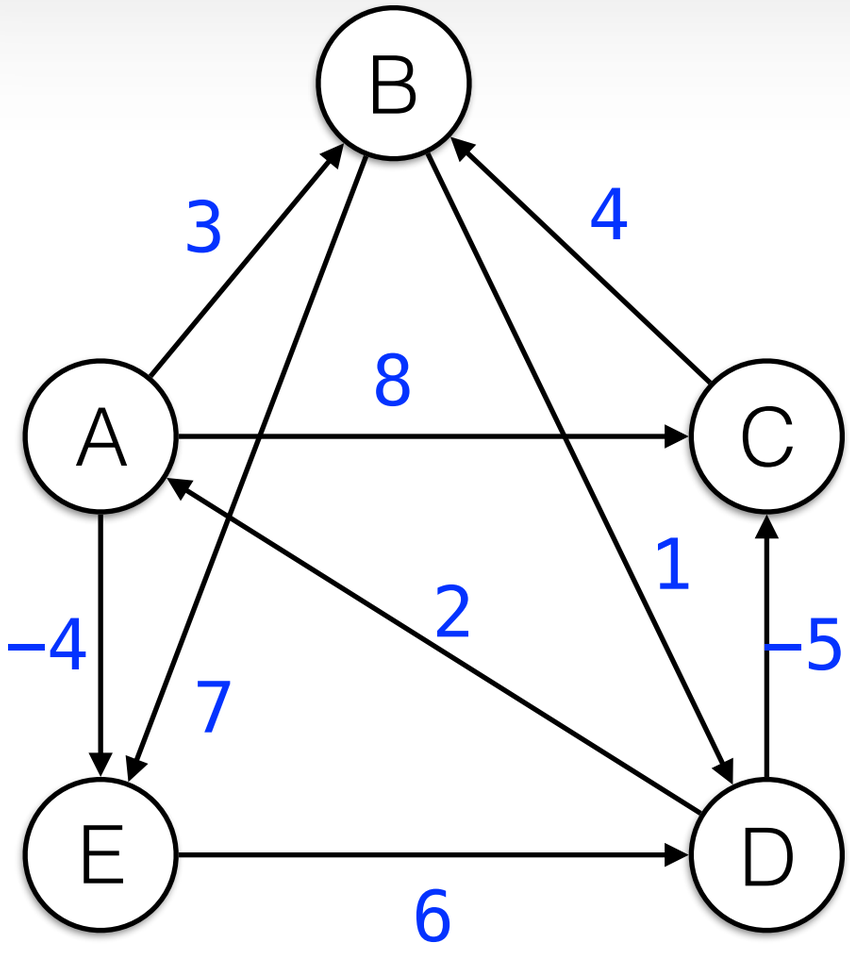
D^0	A	B	C	D	E
A	0	3	8	INF	-4
B	INF	0	INF	1	7
C	INF	4	0	INF	INF
D	2	INF	-5	0	INF
E	INF	INF	INF	6	0

Floyd-Warshall



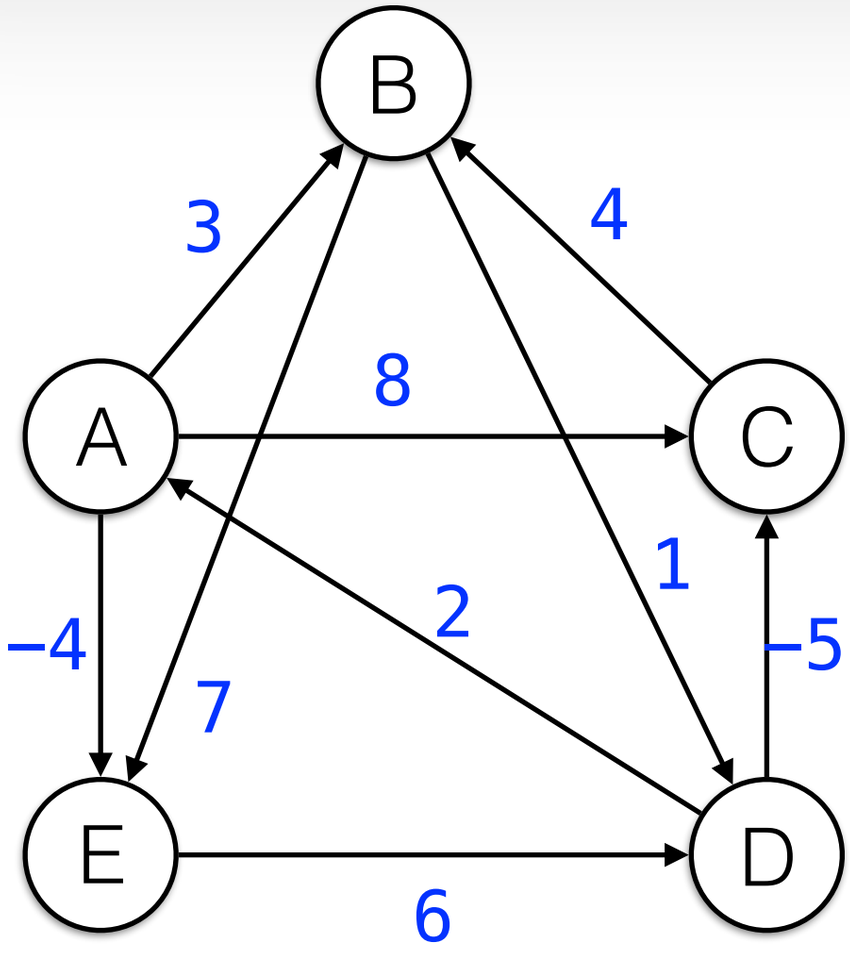
D^1	A	B	C	D	E
A	**0**	**3**	**8**	**INF**	**-4**
B	**INF**	0			
C	**INF**		0		
D	**2**			0	
E	**INF**				0

Floyd-Warshall



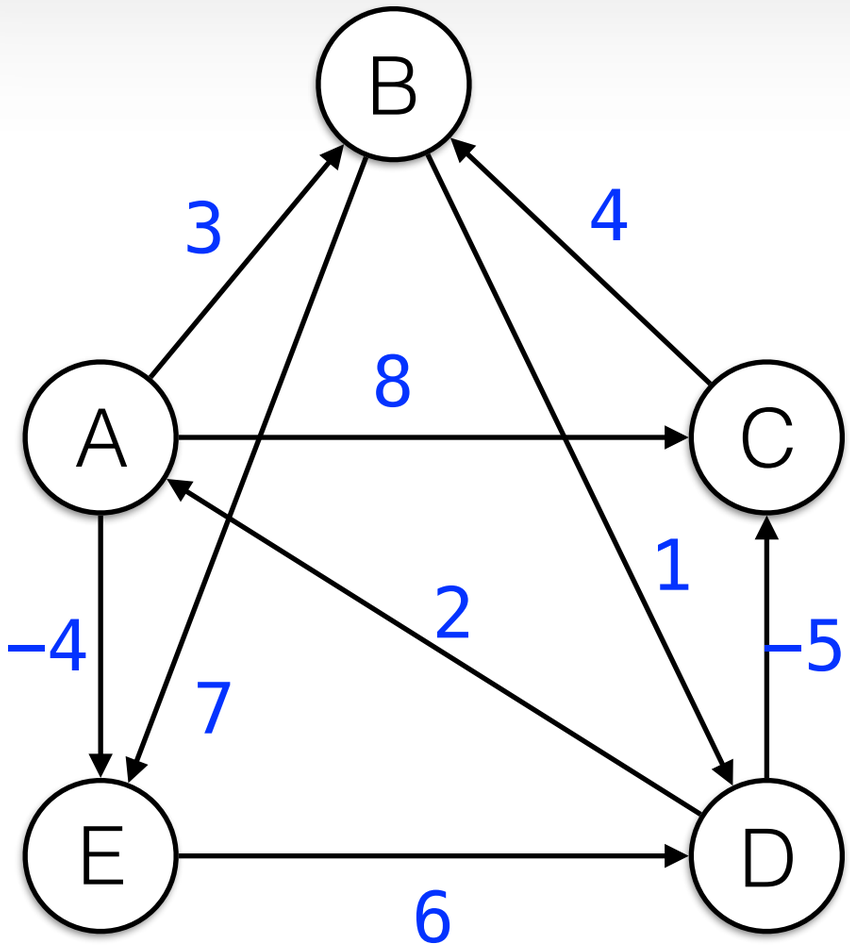
D^1	A	B	C	D	E
A	**0**	**3**	**8**	**INF**	**-4**
B	**INF**	**0**	**INF**	**1**	**7**
C	**INF**	4	0	INF	INF
D	**2**			0	
E	**INF**				0

Floyd-Warshall



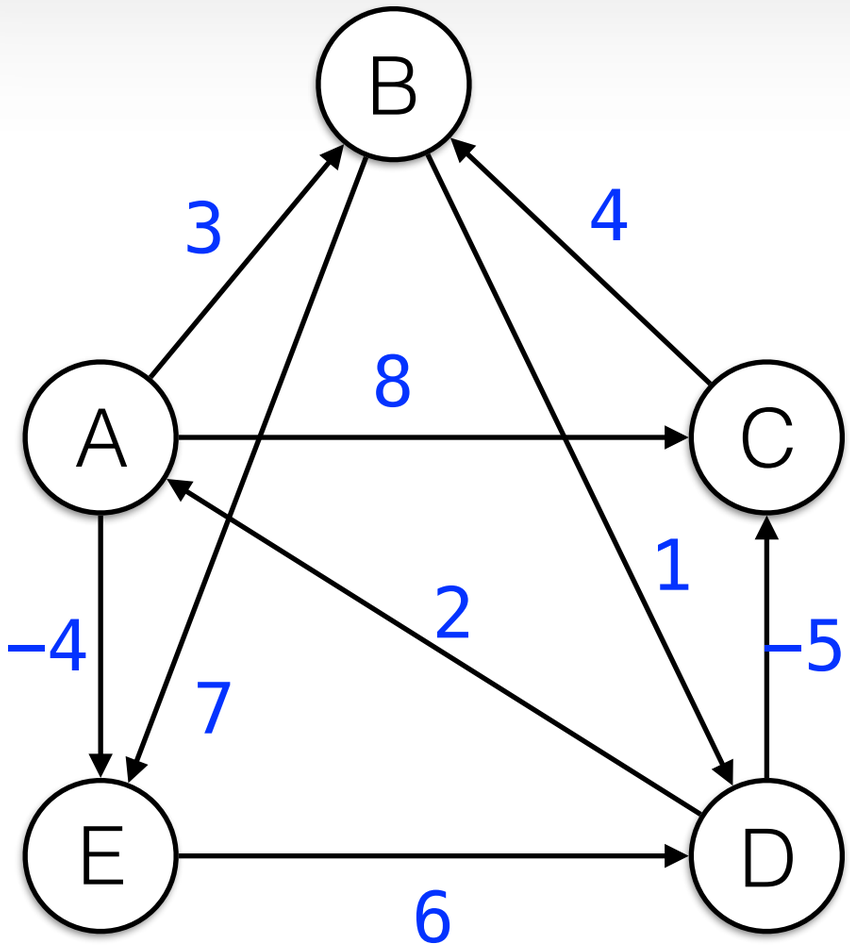
D^1	A	B	C	D	E
A	**0**	**3**	**8**	**INF**	**-4**
B	**INF**	0	INF	1	7
C	**INF**	**4**	**0**	**INF**	**INF**
D	**2**			0	
E	**INF**				0

Floyd-Warshall



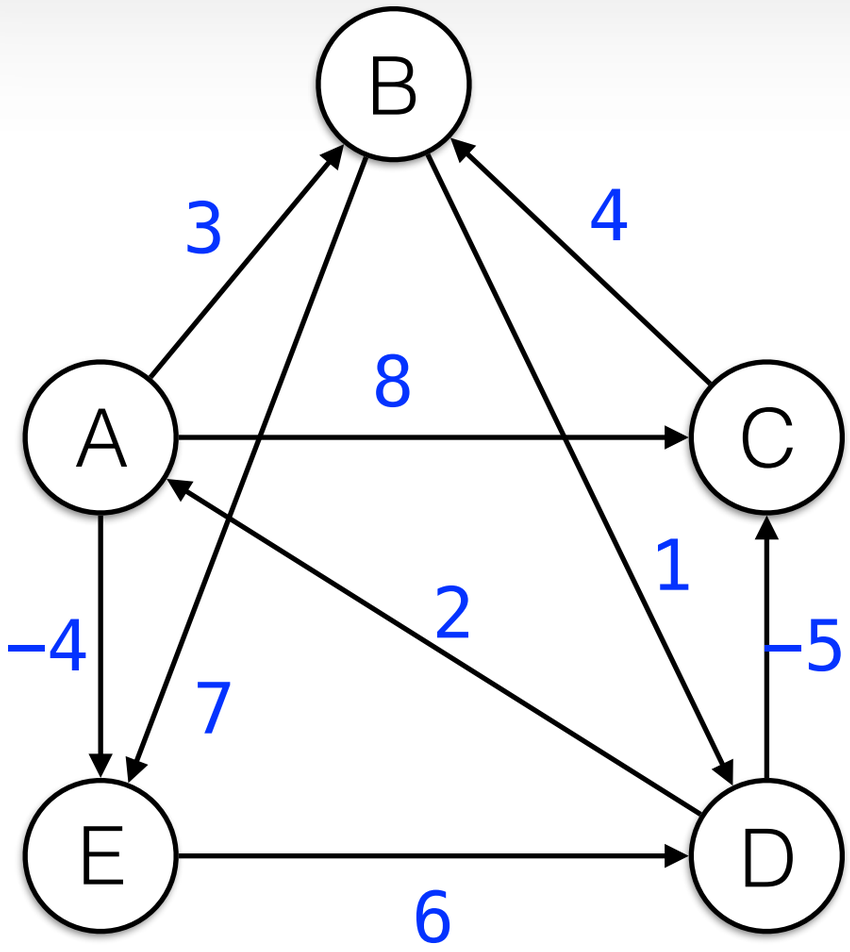
D^1	A	B	C	D	E
A	**0**	**3**	**8**	**INF**	**-4**
B	**INF**	0	INF	1	7
C	**INF**	4	0	INF	INF
D	**2**	**5**	**-5**	0	**-2**
E	**INF**				0

Floyd-Warshall



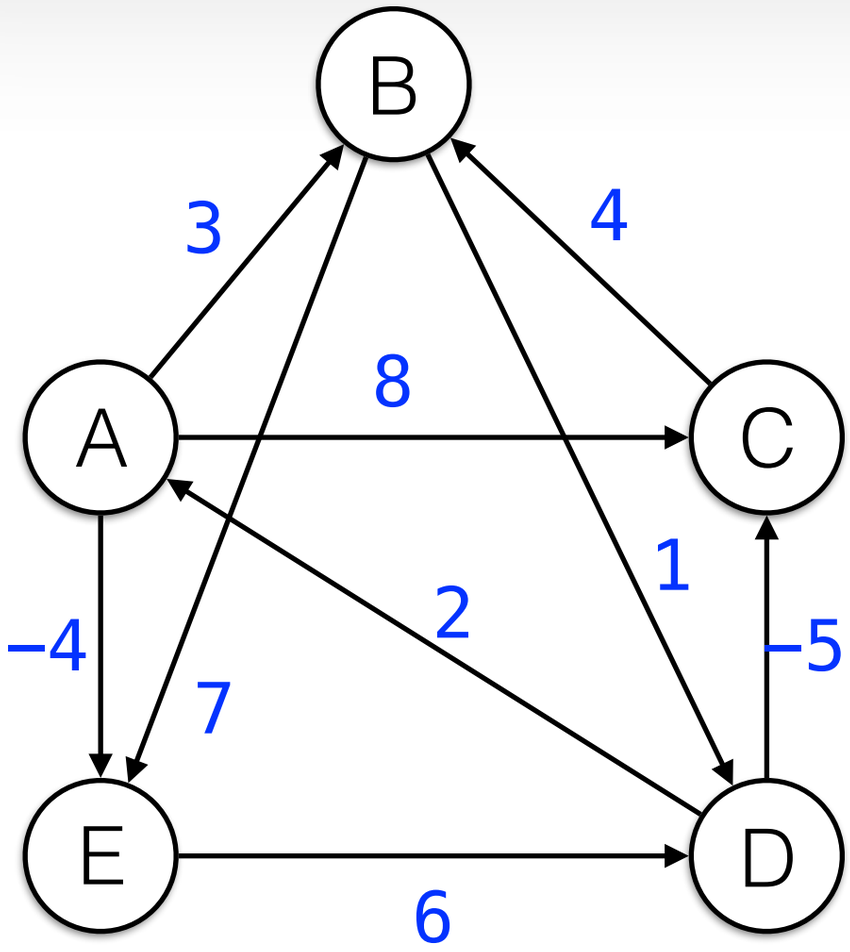
D^1	A	B	C	D	E
A	**0**	**3**	**8**	**INF**	**-4**
B	**INF**	0	INF	1	7
C	**INF**	4	0	INF	INF
D	**2**	5	-5	0	-2
E	**INF**	**INF**	**INF**	**6**	0

Floyd-Warshall



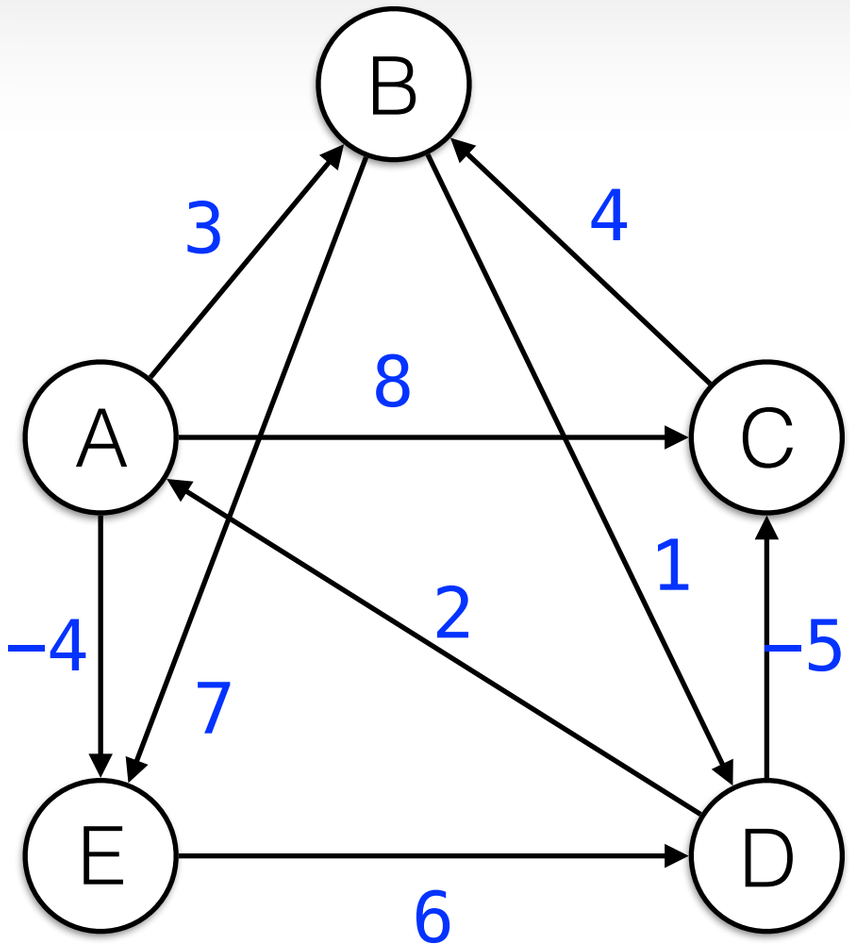
D^2	A	B	C	D	E
A	0	**3**	8	INF	-4
B	**INF**	**0**	**INF**	**1**	**7**
C	INF	**4**	0	INF	INF
D	2	**5**	-5	0	-2
E	INF	**INF**	INF	6	0

Floyd-Warshall



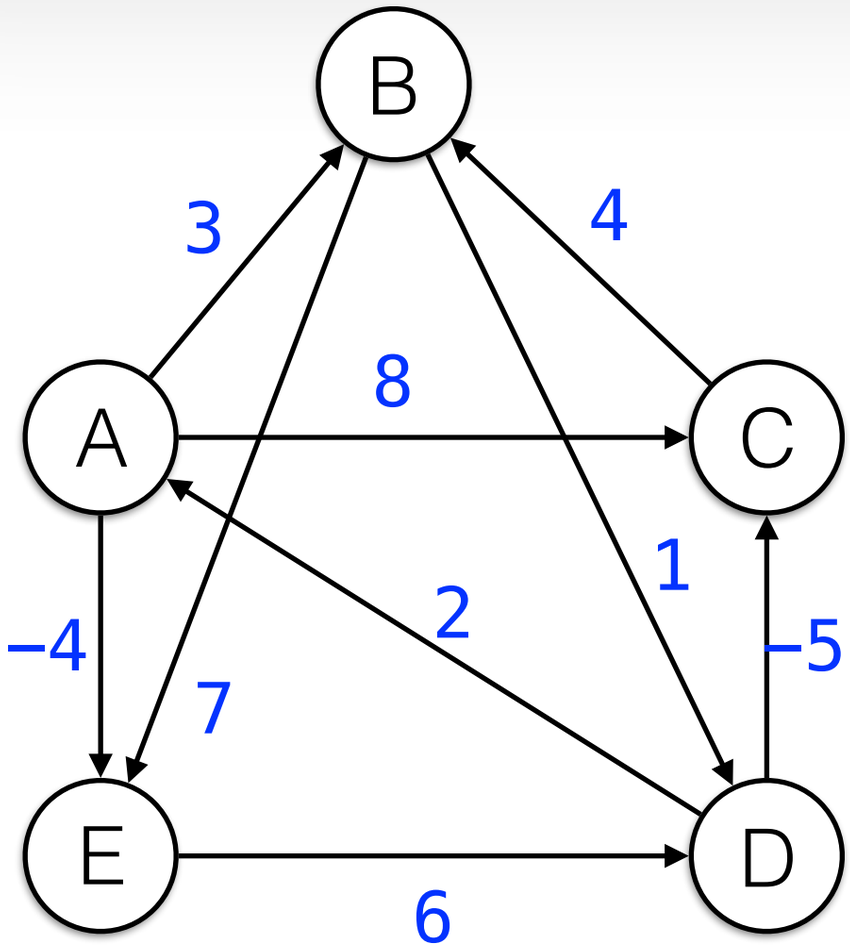
D^2	A	B	C	D	E
A	0	**3**	8	**4**	**-4**
B	**INF**	**0**	**INF**	**1**	**7**
C	INF	**4**	0	INF	INF
D	2	**5**	-5	0	-2
E	INF	**INF**	INF	6	0

Floyd-Warshall



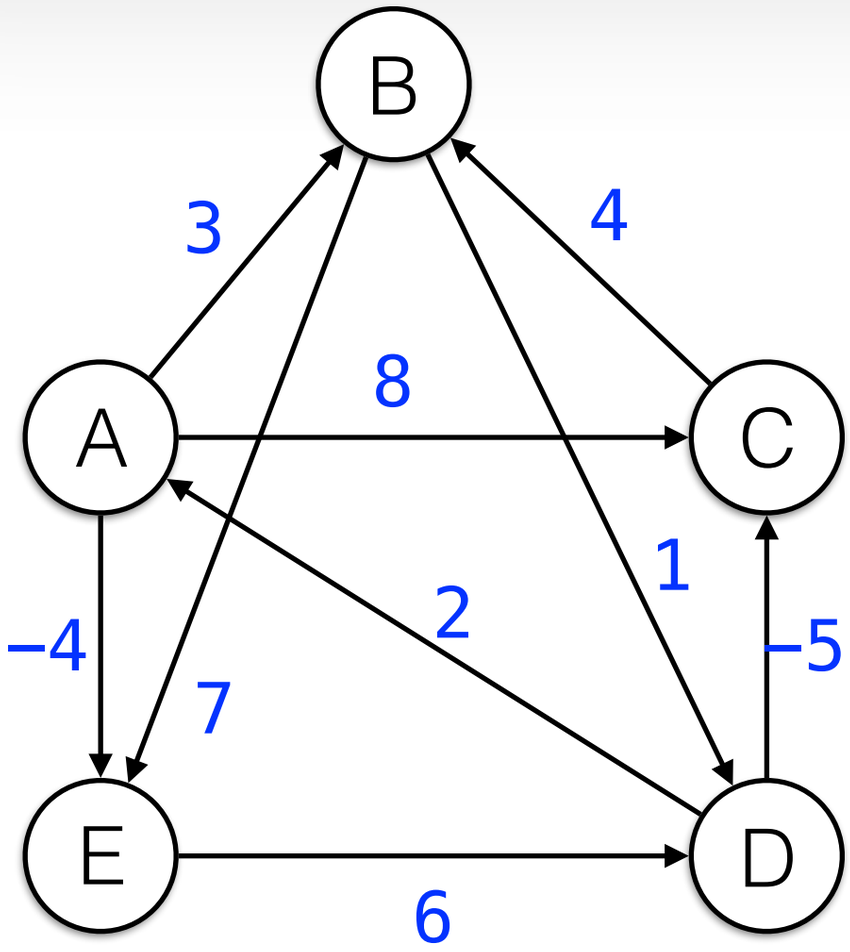
D^2	A	B	C	D	E
A	0	**3**	8	4	-4
B	**INF**	**0**	**INF**	**1**	**7**
C	INF	**4**	0	**5**	**11**
D	2	**5**	-5	0	-2
E	INF	**INF**	INF	6	0

Floyd-Warshall



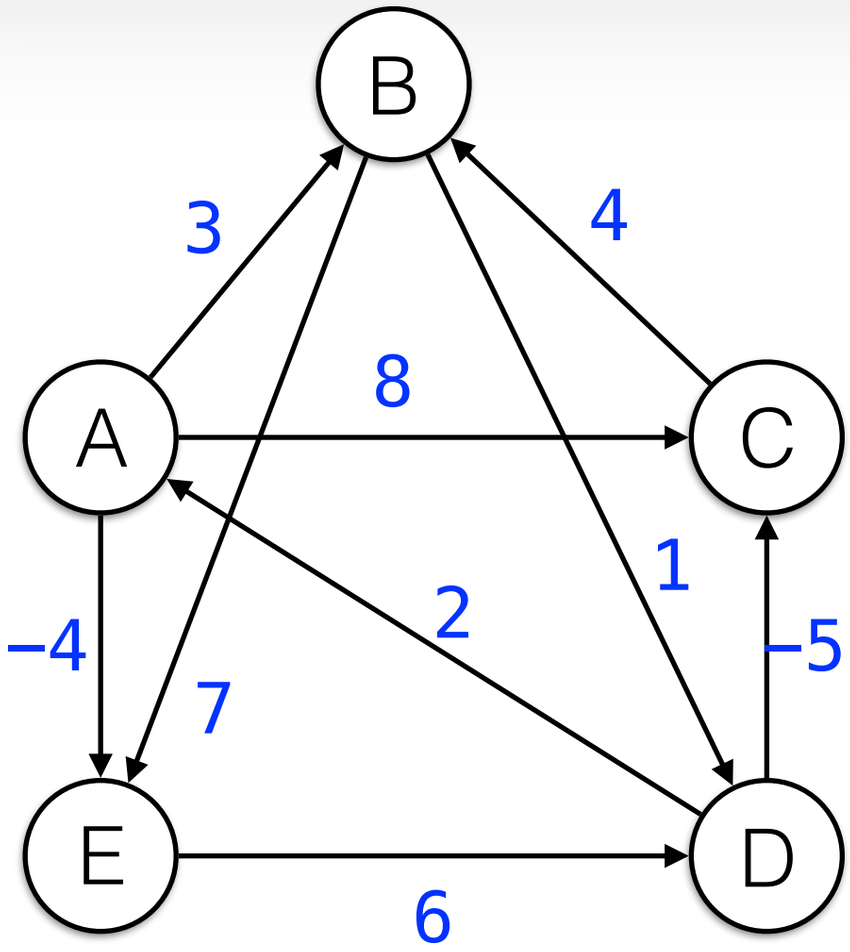
D^2	A	B	C	D	E
A	0	**3**	8	4	-4
B	**INF**	**0**	**INF**	**1**	**7**
C	INF	**4**	0	5	11
D	2	**5**	-5	**0**	**-2**
E	INF	**INF**	INF	6	0

Floyd-Warshall



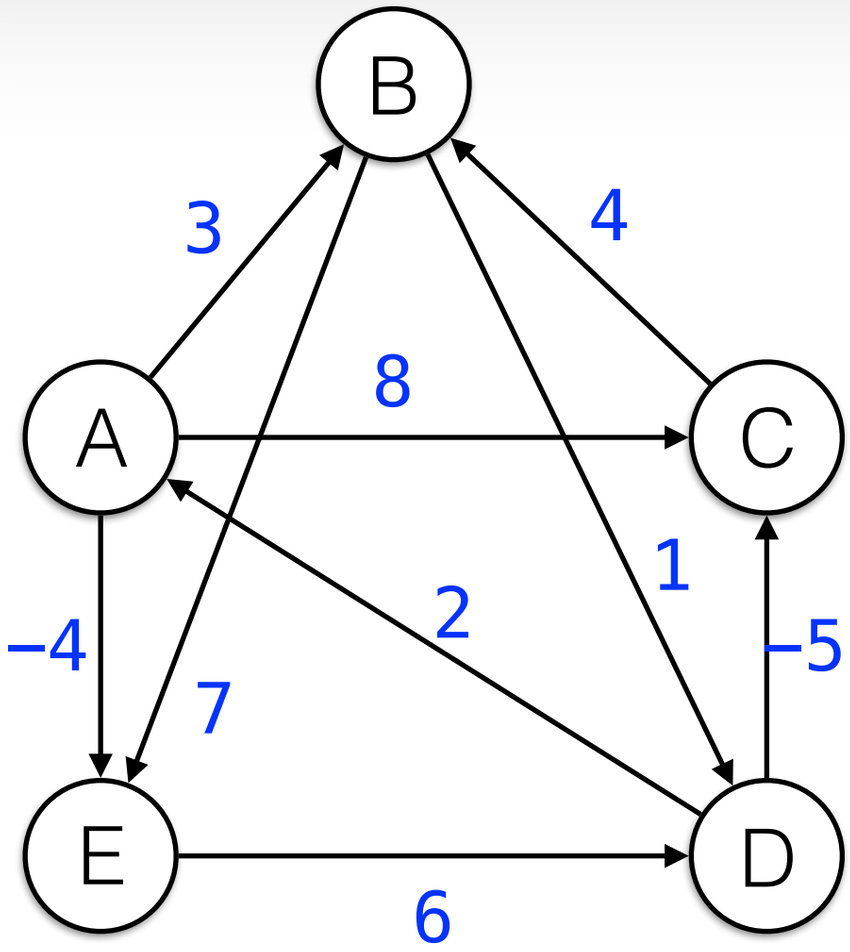
D^2	A	B	C	D	E
A	0	**3**	8	4	-4
B	**INF**	**0**	**INF**	**1**	**7**
C	INF	**4**	0	5	11
D	2	**5**	-5	0	-2
E	INF	**INF**	INF	6	0

Floyd-Warshall



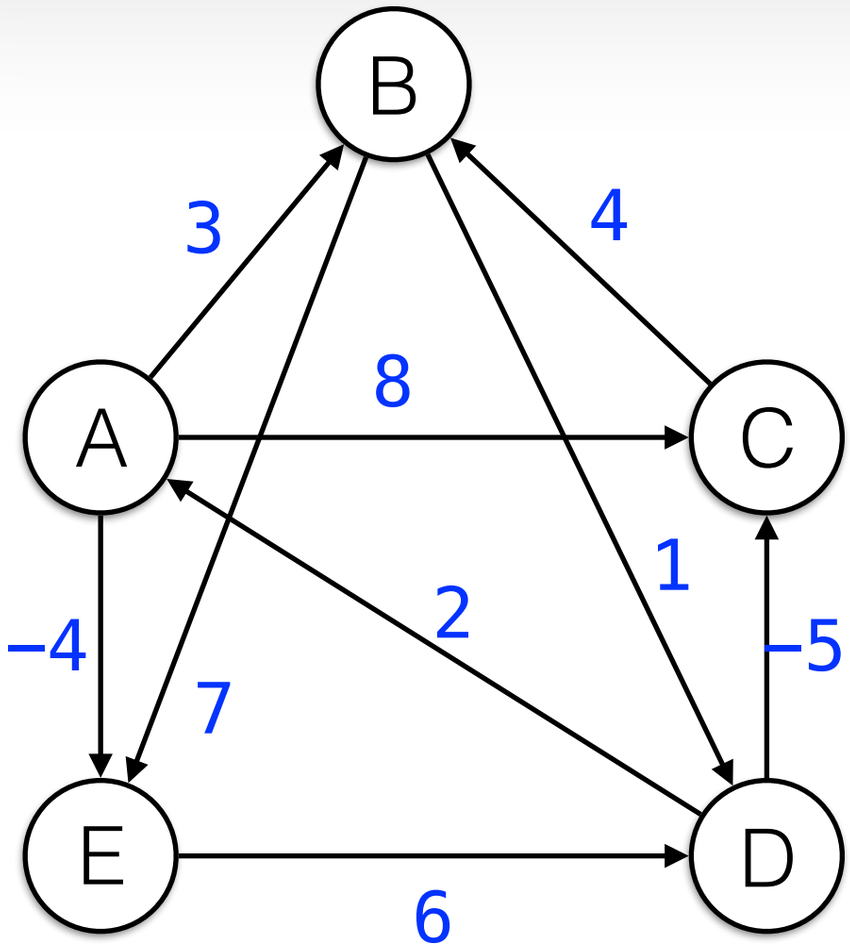
D^3	A	B	C	D	E
A	0	**3**	**8**	**4**	**-4**
B	INF	0	**INF**	1	7
C	**INF**	**4**	**0**	**5**	**11**
D	2	5	**-5**	0	-2
E	INF	INF	**INF**	6	0

Floyd-Warshall



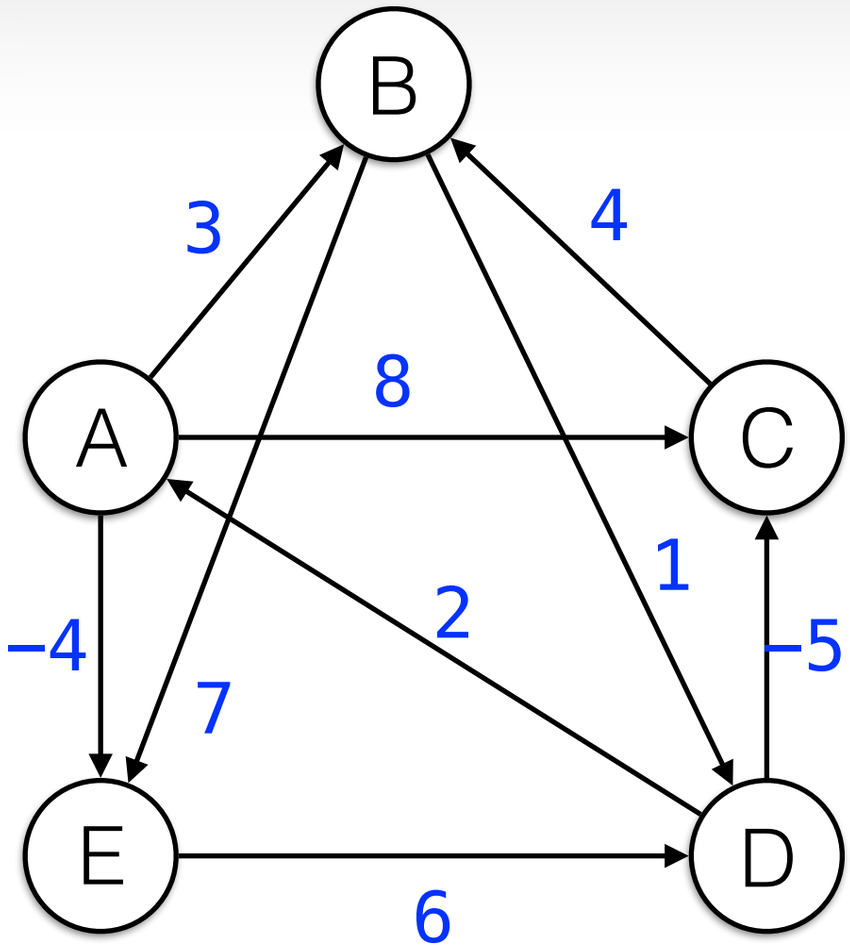
D^3	A	B	C	D	E
A	0	3	**8**	4	-4
B	INF	0	**INF**	1	7
C	**INF**	**4**	**0**	**5**	**11**
D	2	**-1**	**-5**	**0**	**-2**
E	INF	INF	**INF**	6	0

Floyd-Warshall



D^4	A	B	C	D	E
A	0	3	-1	**4**	-4
B	3	0	-4	**1**	-1
C	7	4	0	**5**	3
D	**2**	**-1**	**-5**	**0**	**-2**
E	8	5	1	**6**	0

Floyd-Warshall



D^5	A	B	C	D	E
A	0	0	1	-3	**-4**
B	3	0	-4	1	**-1**
C	7	4	0	5	**3**
D	2	-1	-5	0	**-2**
E	**8**	**5**	**1**	**6**	**0**

Bellman-Ford

Objective

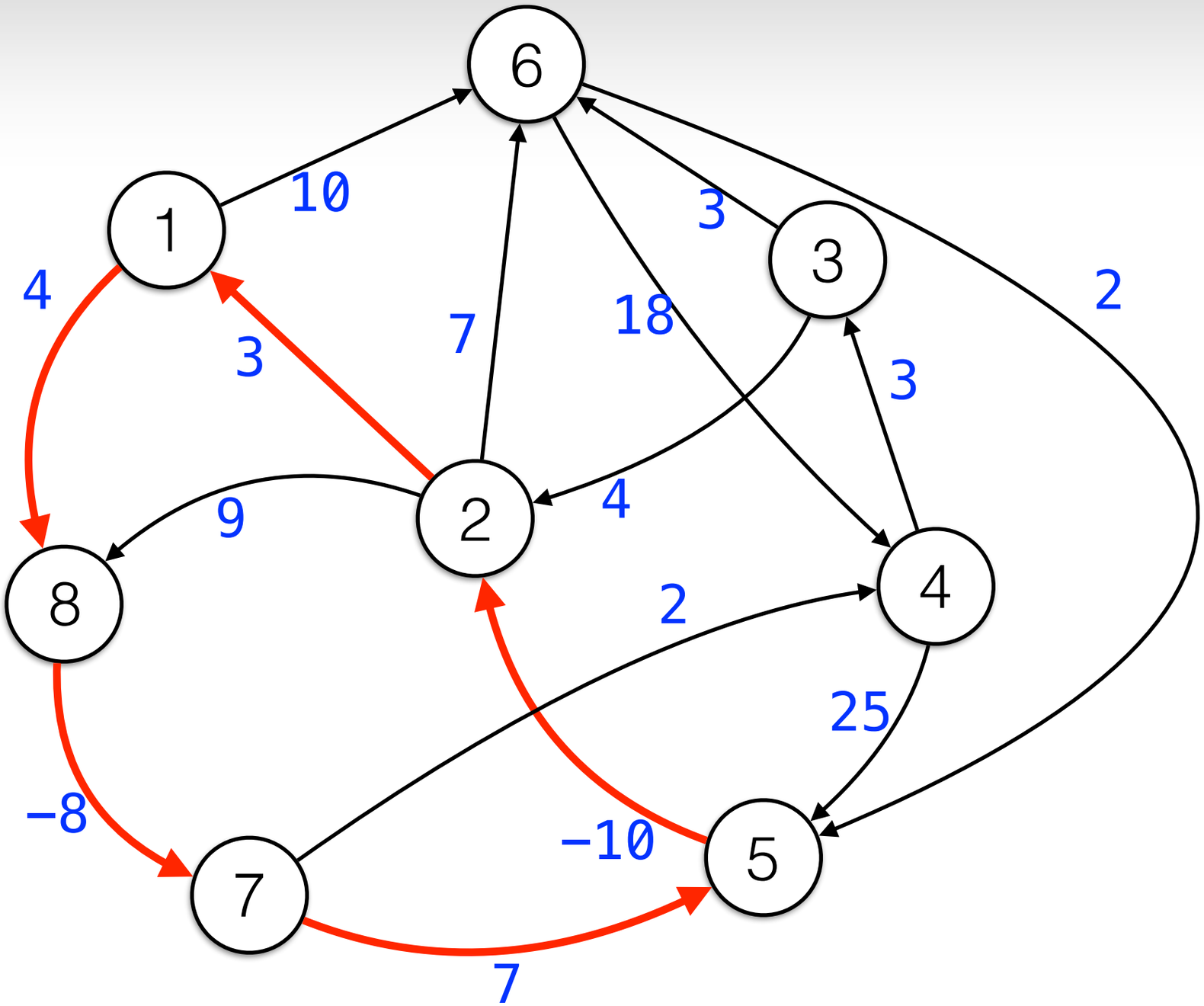
DP algorithm to find the shortest path between two nodes, of a particular degree.

Can have negative arcs, it finds negative cycles

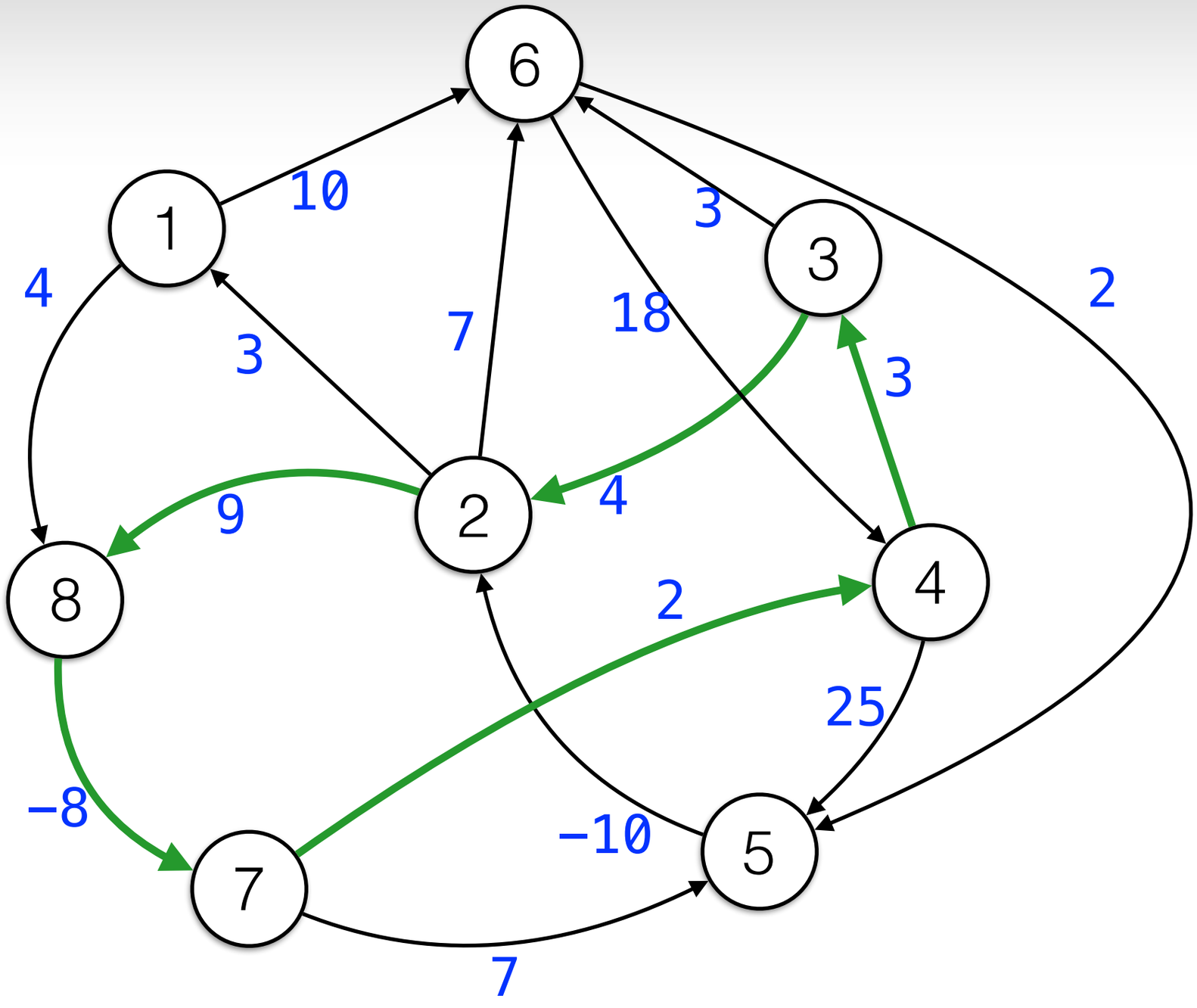
How to

For a node `s`, build a tree (Dijkstra) with the shortest path to every other node.

ciclos negativos



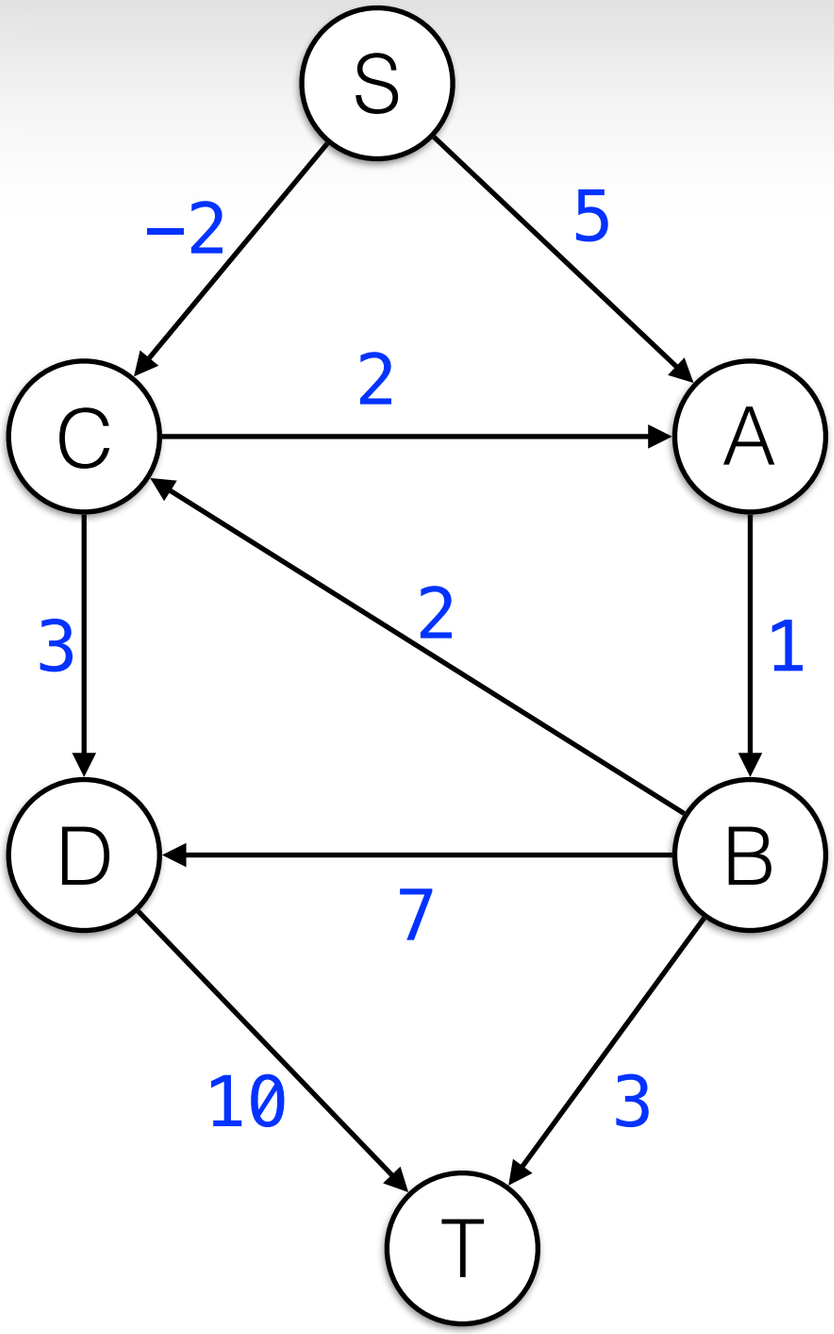
ciclos negativos



Bellman-Ford

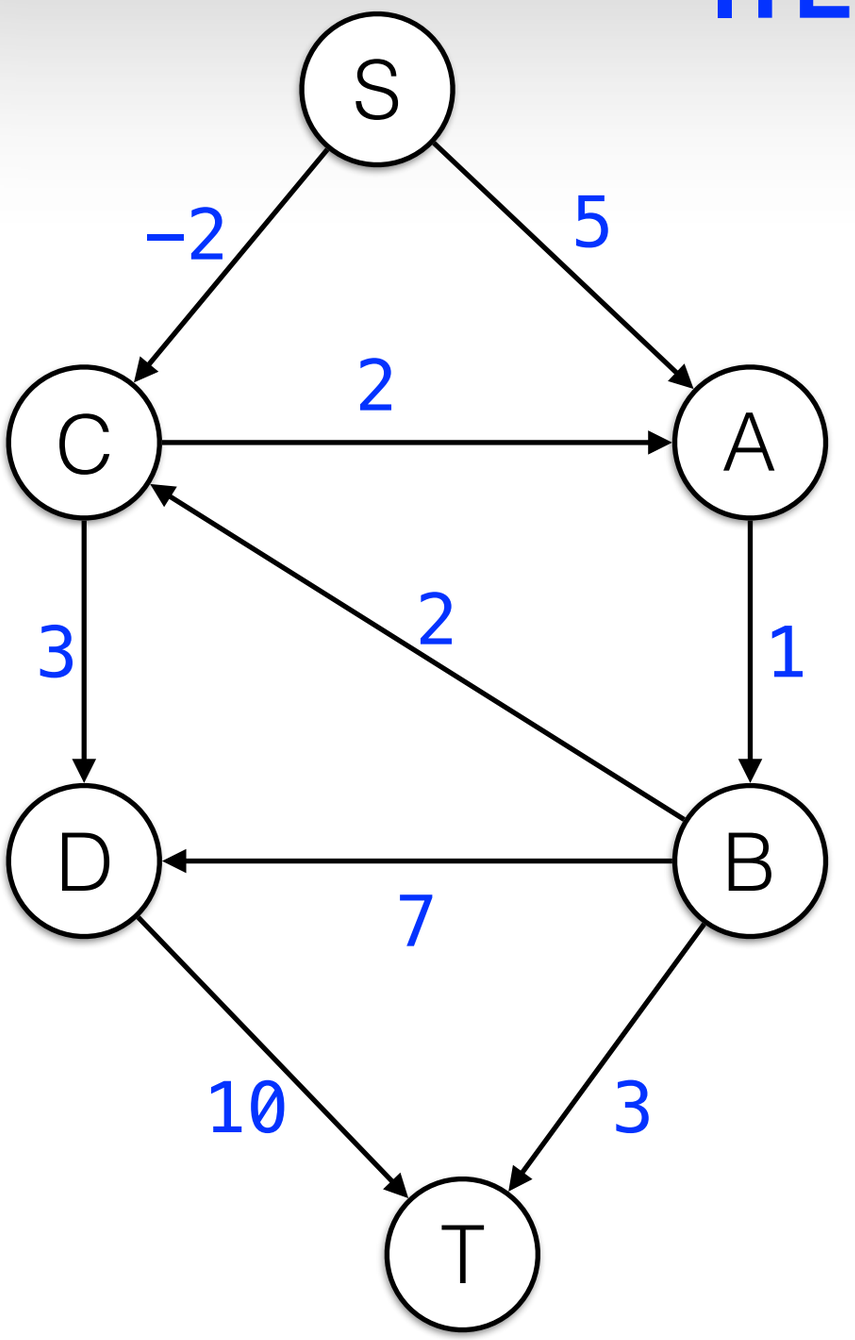
```
public void bellmanFord(V s) {
 Double d[G.vertices().size()]={Double.POSITIVE_INFINITY};
 V pi[G.vertices().size()]={null};
 d[s] = 0;
 for(int i=0; i<G.vertices().size()-1; i++) {
  for(E e : G.edges()) {
   relax(e.source(), e.dest());
  }
 }
 for(E e : G.edges()) {
   if(d[e.dest()] > d[e.source()] + e.weight() return false;
 }
 return true;
}
```

Bellman-Ford



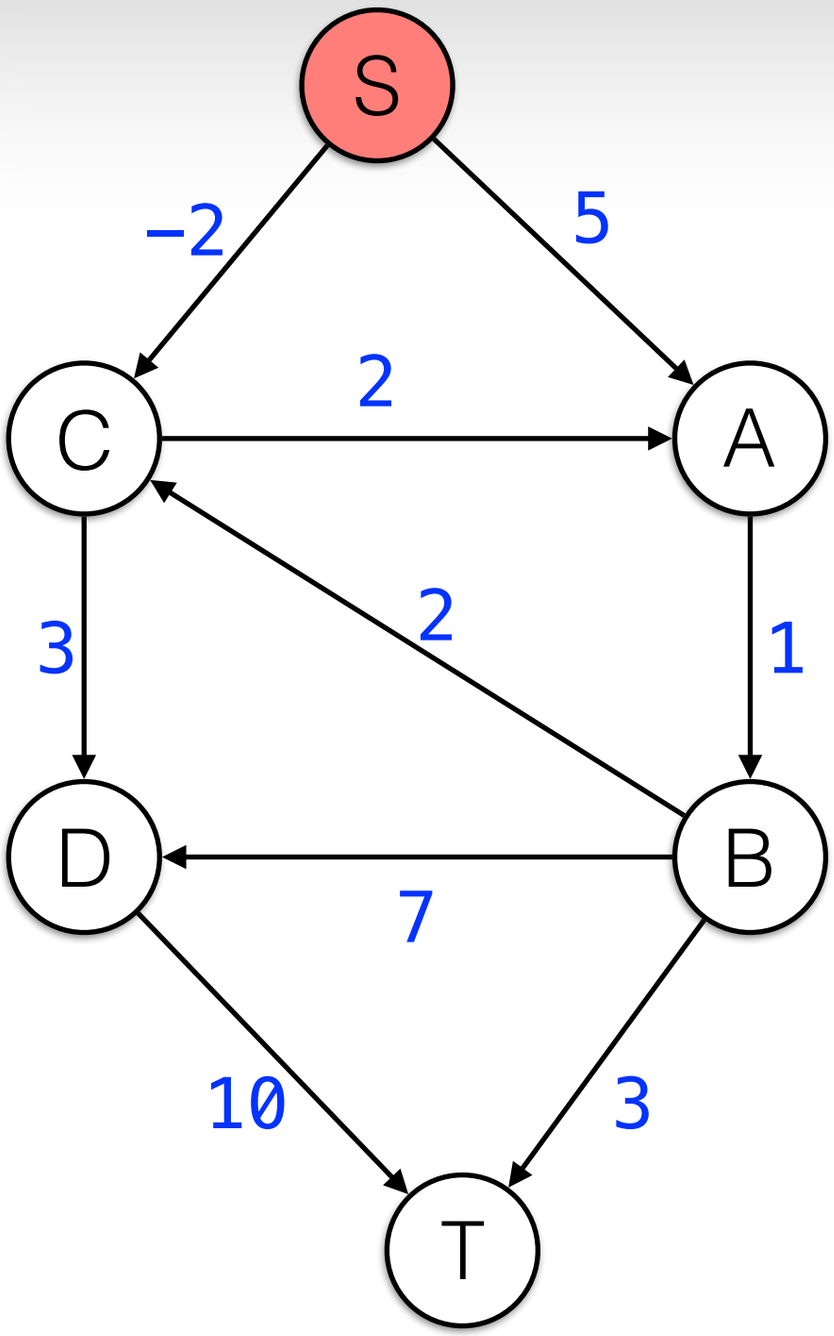
Bellman-Ford

ITERACIÓN 1



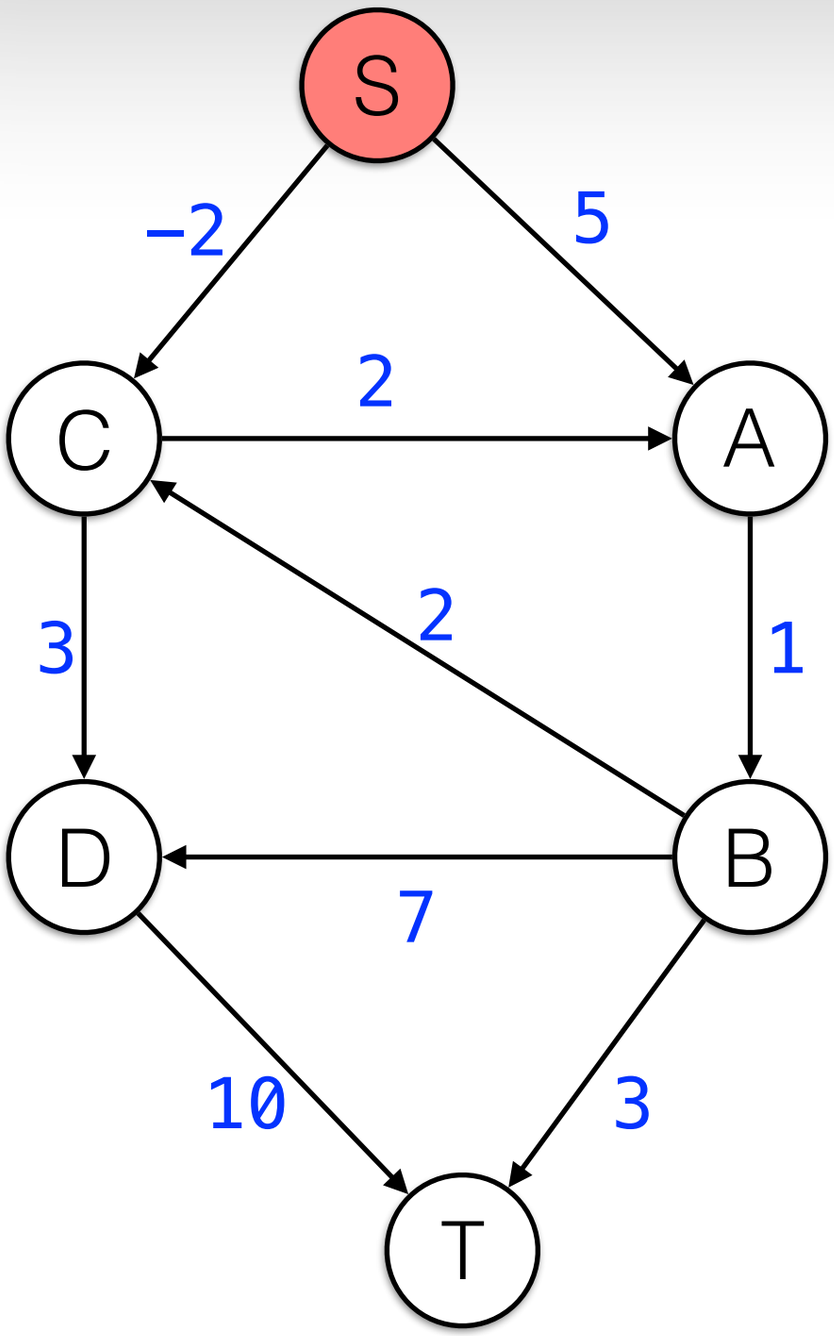
v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
S	0	null
A	INFINITY	null
B	INFINITY	null
C	INFINITY	null
D	INFINITY	null
T	INFINITY	null

Bellman-Ford



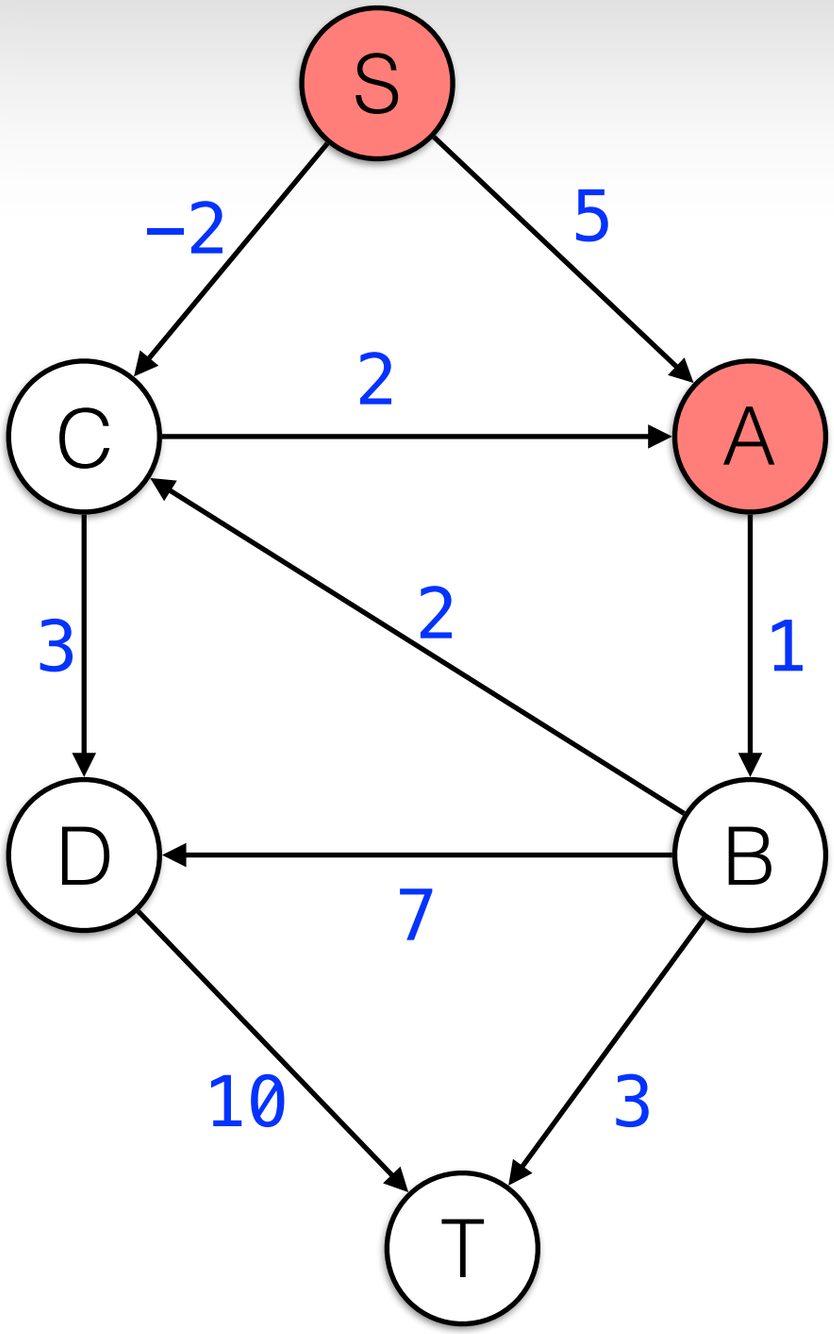
v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
S	0	null
A	**5**	**S**
B	INFINITY	null
C	INFINITY	null
D	INFINITY	null
T	INFINITY	null

Bellman-Ford



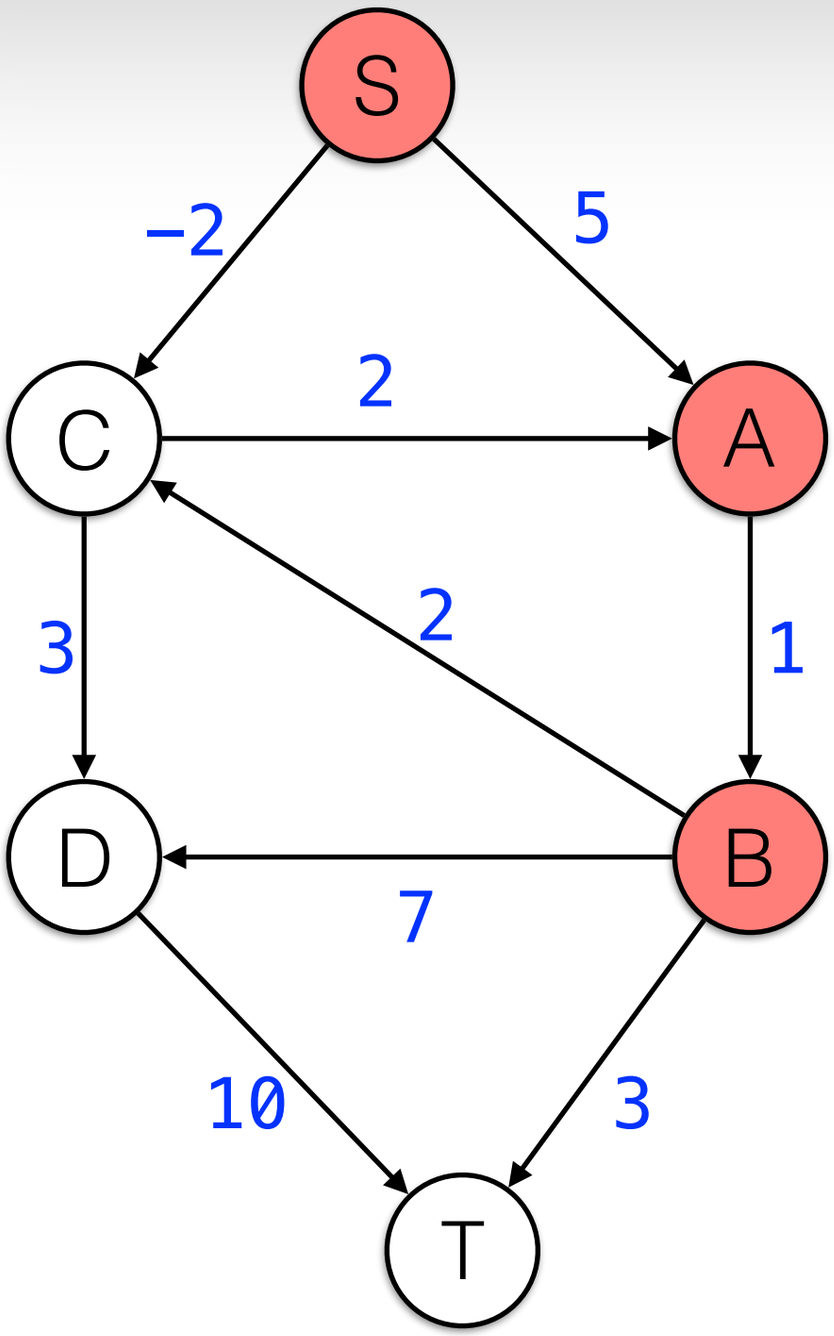
v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
S	0	null
A	5	S
B	INFINITY	null
C	**-2**	**S**
D	INFINITY	null
T	INFINITY	null

Bellman-Ford



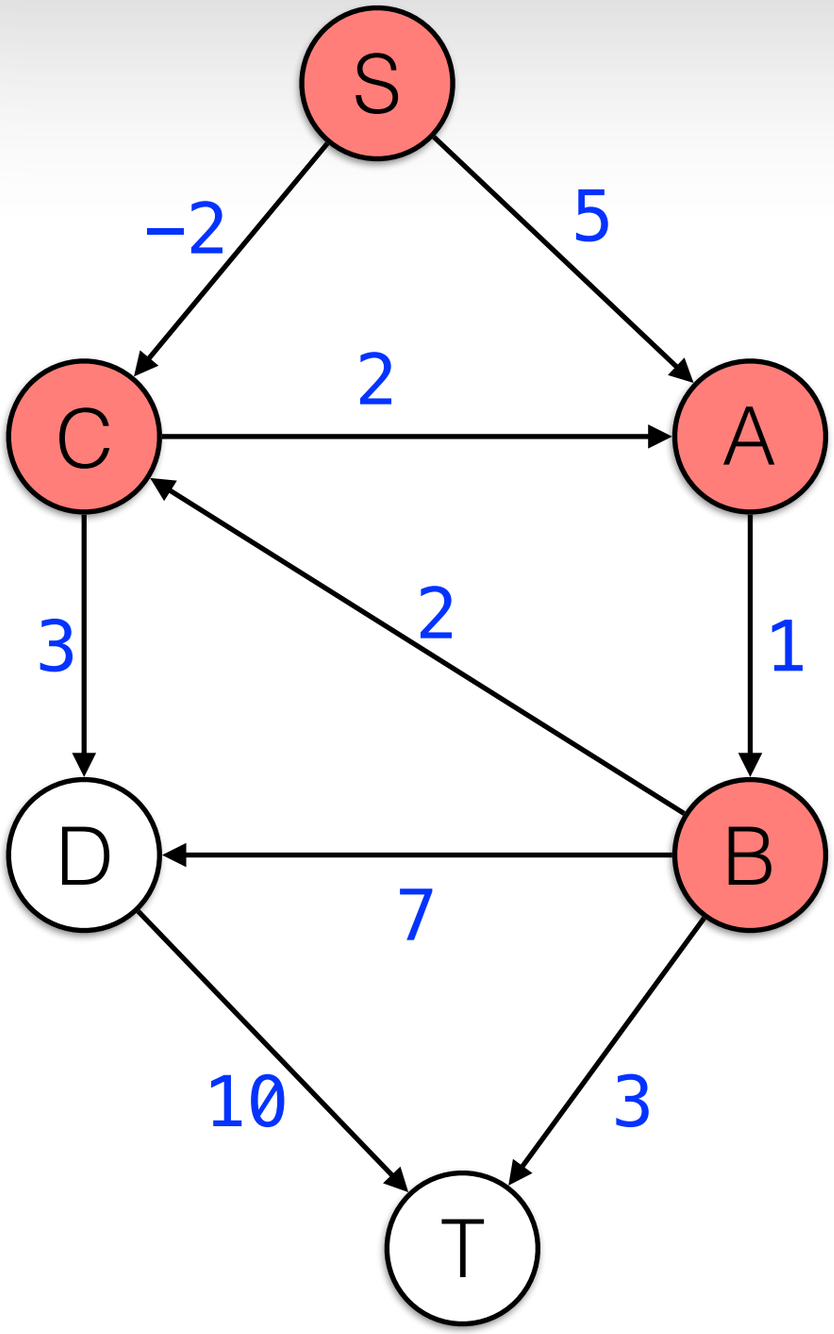
v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
S	0	null
A	5	S
B	**6**	**A**
C	-2	S
D	INFINITY	null
T	INFINITY	null

Bellman-Ford



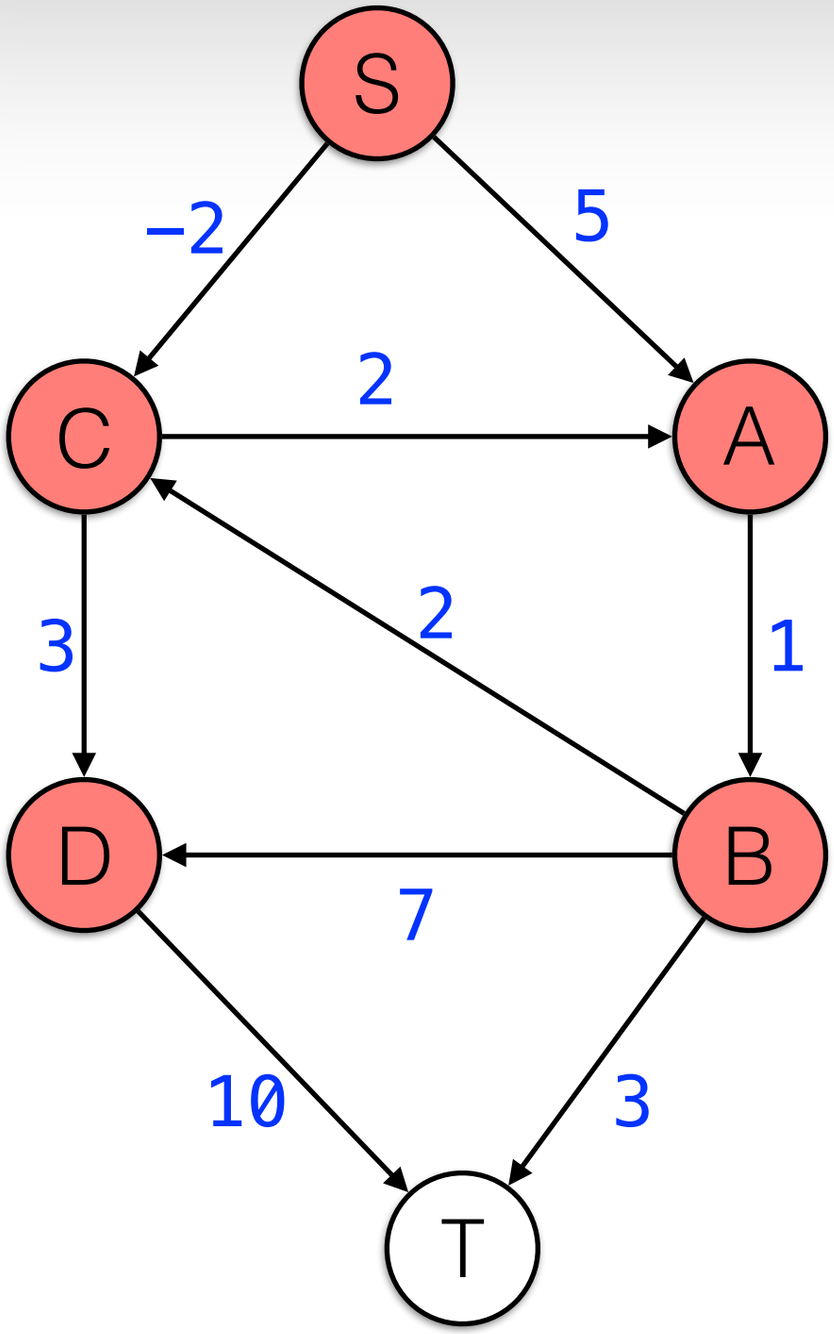
v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
S	0	null
A	5	S
B	6	A
C	**−2**	**S**
D	**13**	**B**
T	**9**	**B**

Bellman-Ford



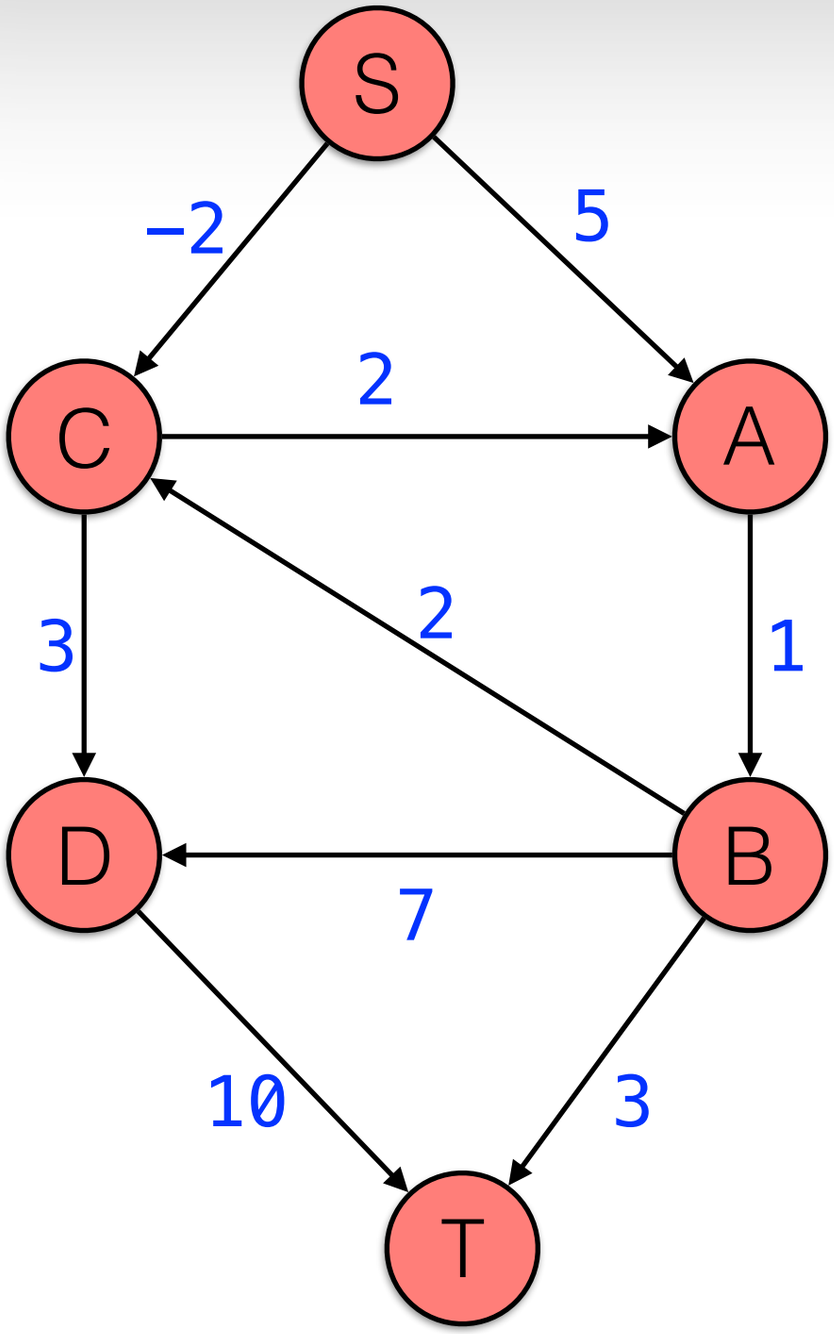
v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
S	0	null
A	**0**	**C**
B	6	A
C	−2	S
D	**1**	**C**
T	9	B

Bellman-Ford



v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
S	0	null
A	0	C
B	6	A
C	-2	S
D	1	C
T	**9**	**B**

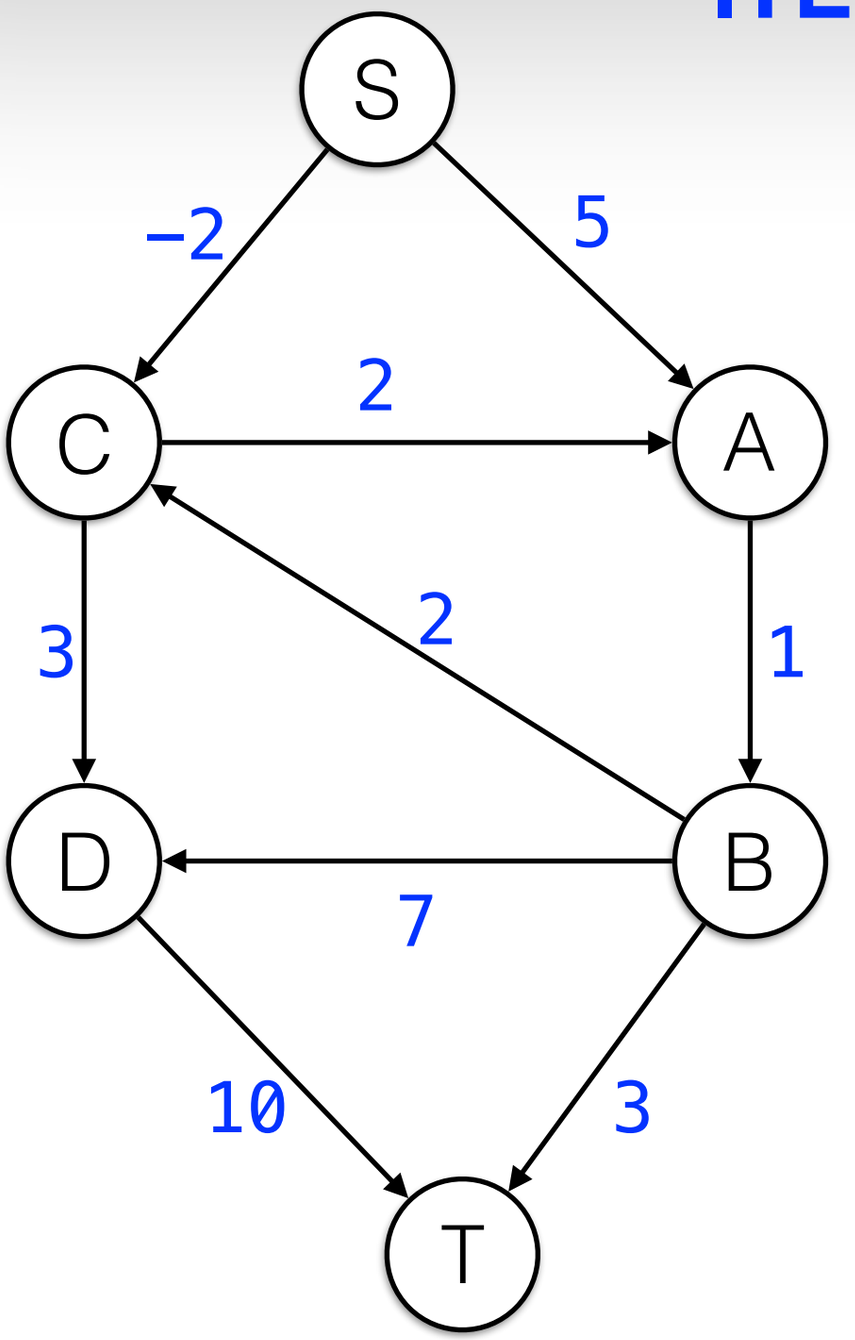
Bellman-Ford



v	distTo[v] $\delta(s,v)$	edge[v] $\pi(v)$
S	0	null
A	0	C
B	6	A
C	-2	S
D	1	C
T	9	B

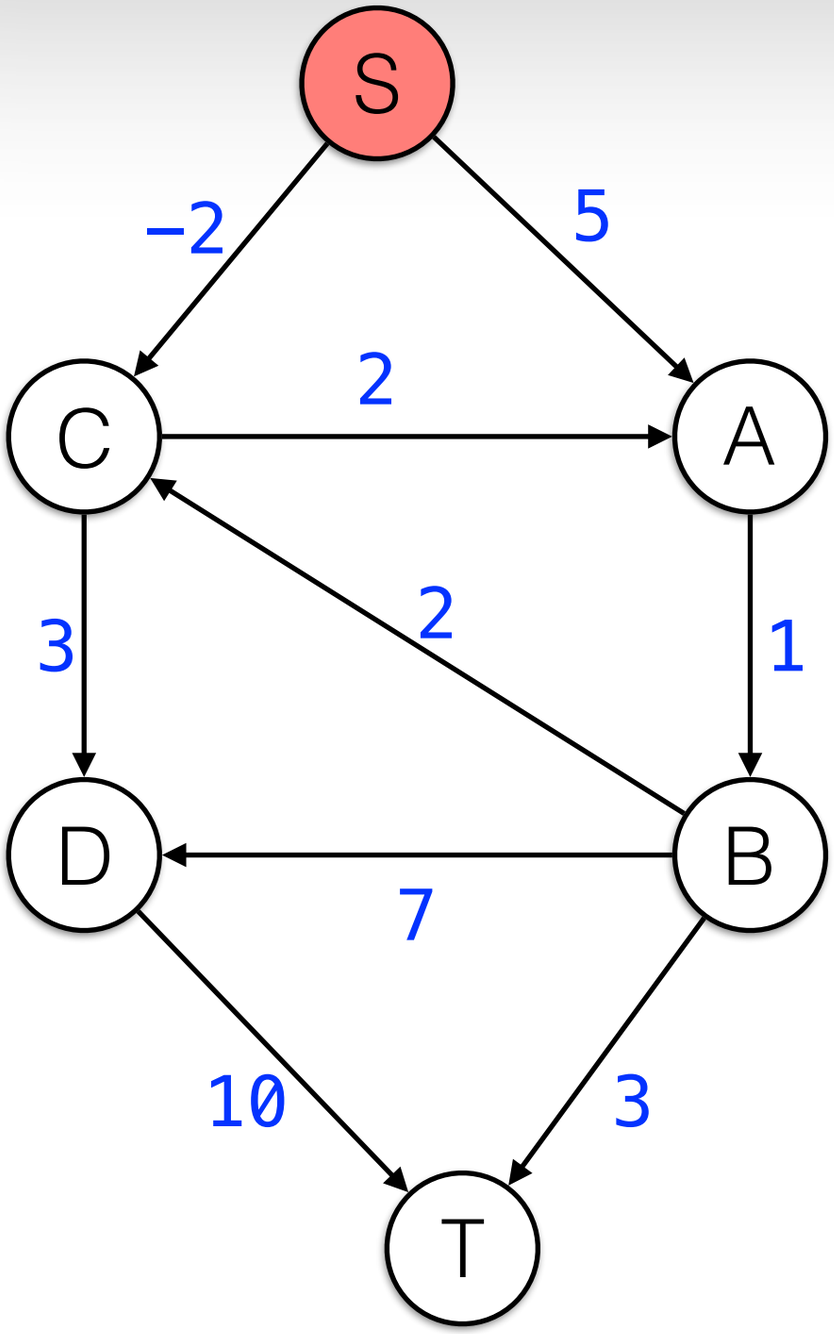
Bellman-Ford

ITERACIÓN 2



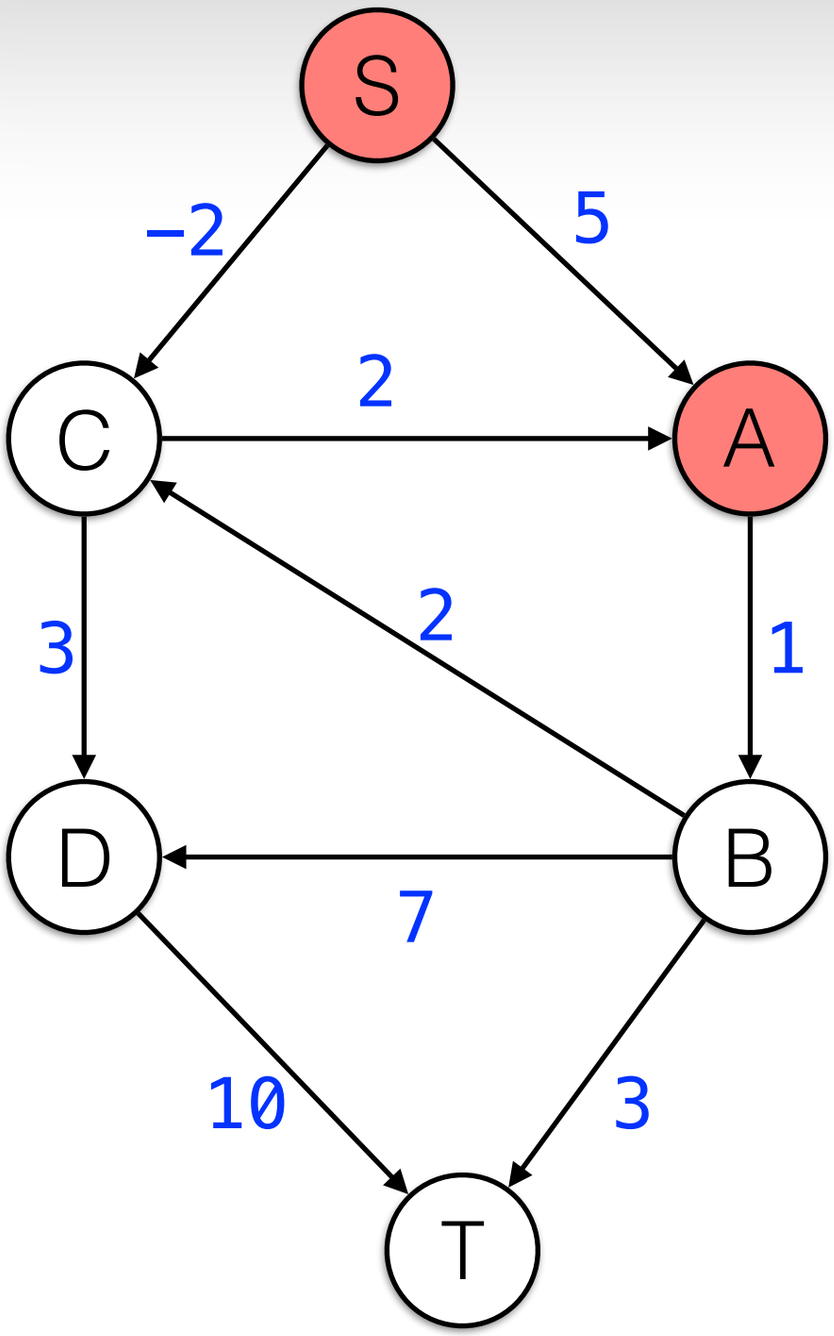
v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
S	0	null
A	0	C
B	6	A
C	-2	S
D	1	C
T	9	B

Bellman-Ford



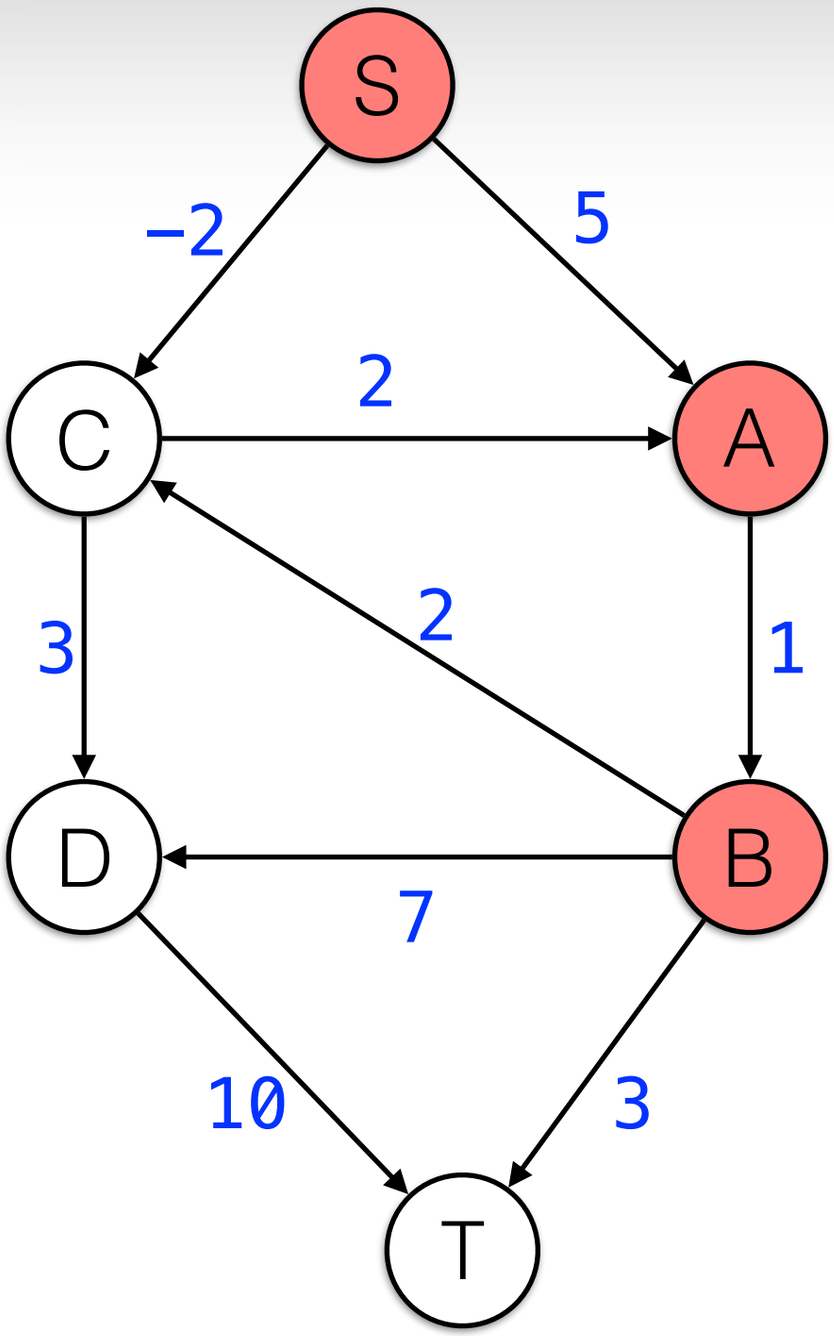
v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
S	0	null
A	**0**	**C**
B	6	A
C	**−2**	**S**
D	1	C
T	9	B

Bellman-Ford



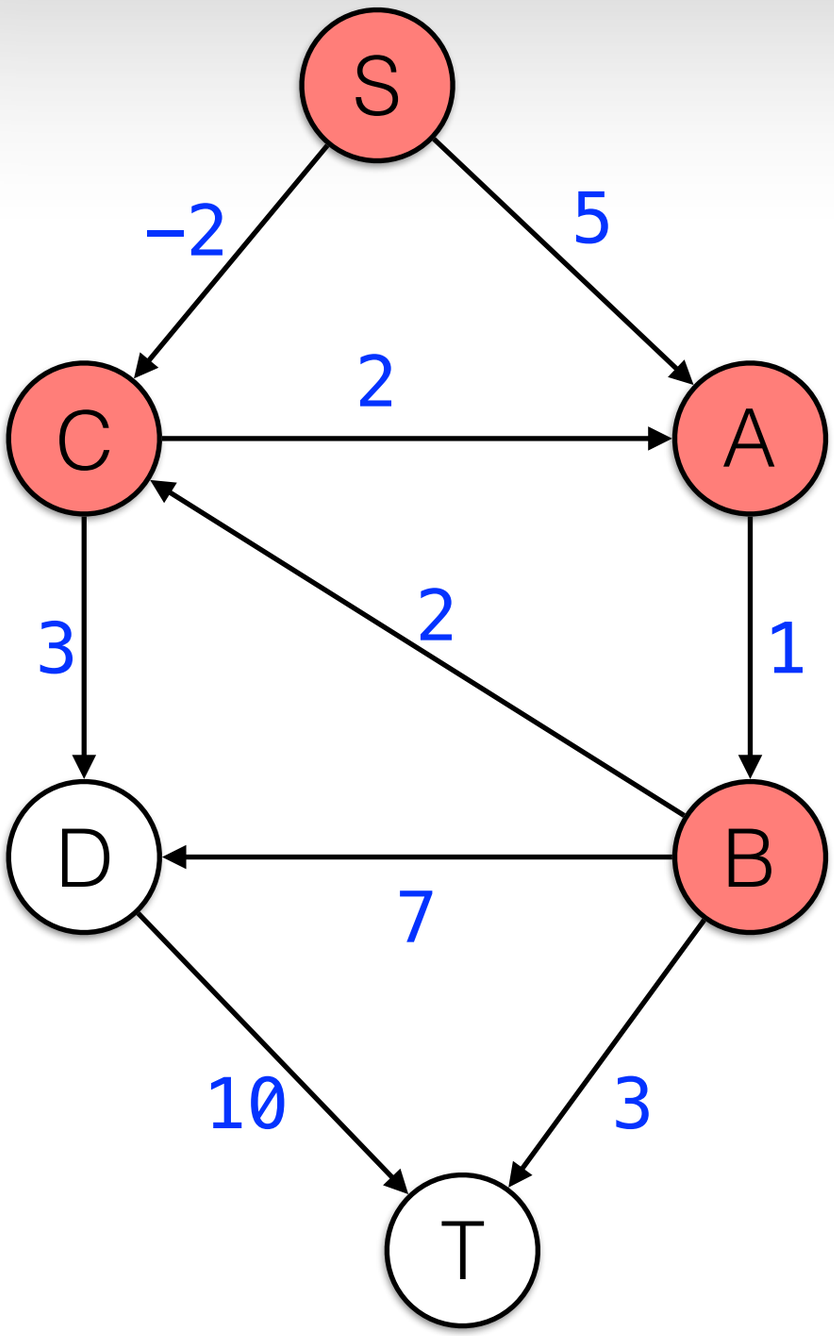
v	distTo[v] $\delta(s,v)$	edge[v] $\pi(v)$
S	0	null
A	0	C
B	**1**	**A**
C	-2	S
D	1	C
T	9	B

Bellman-Ford



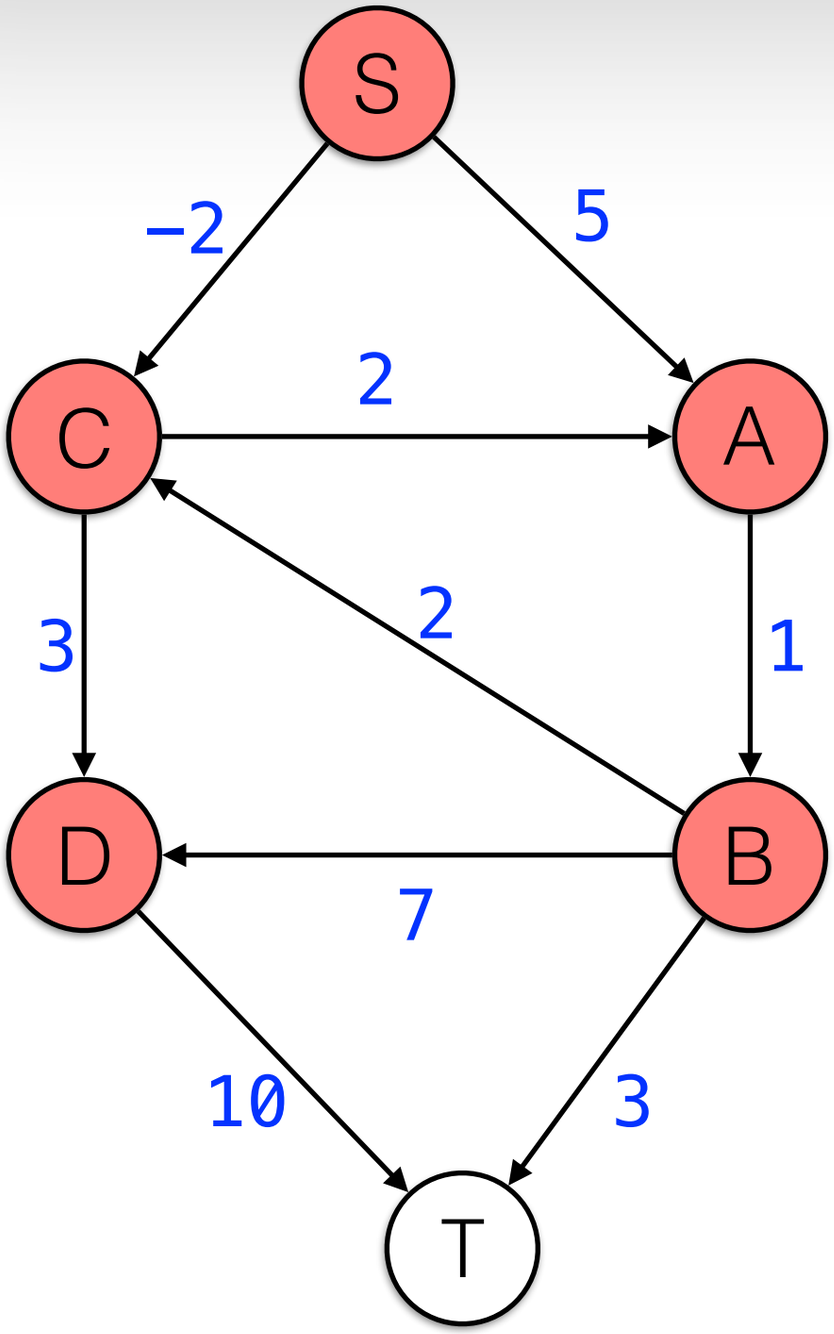
v	distTo[v] $\delta(s,v)$	edge[v] $\pi(v)$
S	0	null
A	0	C
B	1	A
C	-2	S
D	1	C
T	4	B

Bellman-Ford



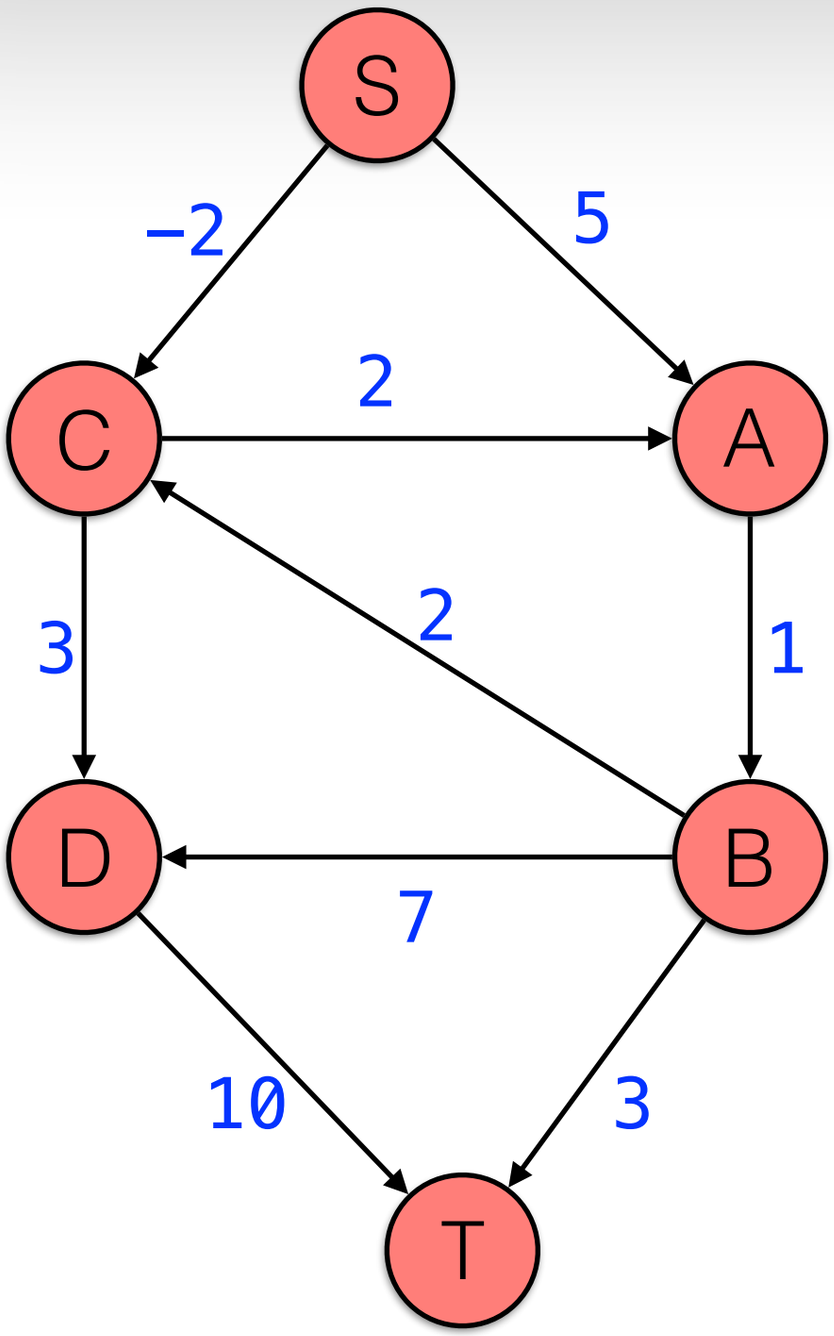
v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
S	0	null
A	**0**	**C**
B	1	A
C	−2	S
D	**1**	**C**
T	4	B

Bellman-Ford



v	distTo[v] $\delta(s,v)$	edge[v] $\pi(v)$
S	0	null
A	0	C
B	1	A
C	-2	S
D	1	C
T	**4**	**B**

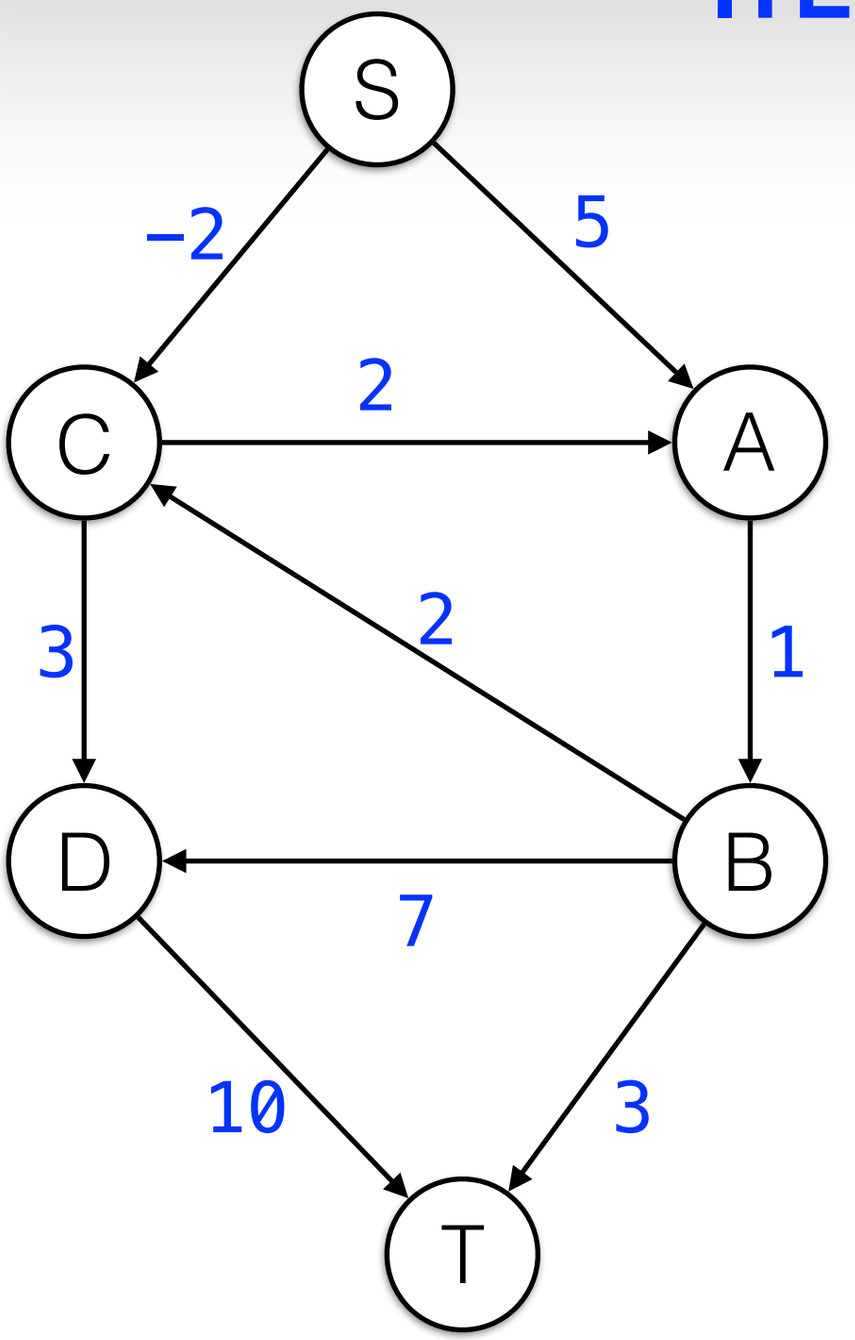
Bellman-Ford



v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
S	0	null
A	0	C
B	1	A
C	-2	S
D	1	C
T	4	B

Bellman-Ford

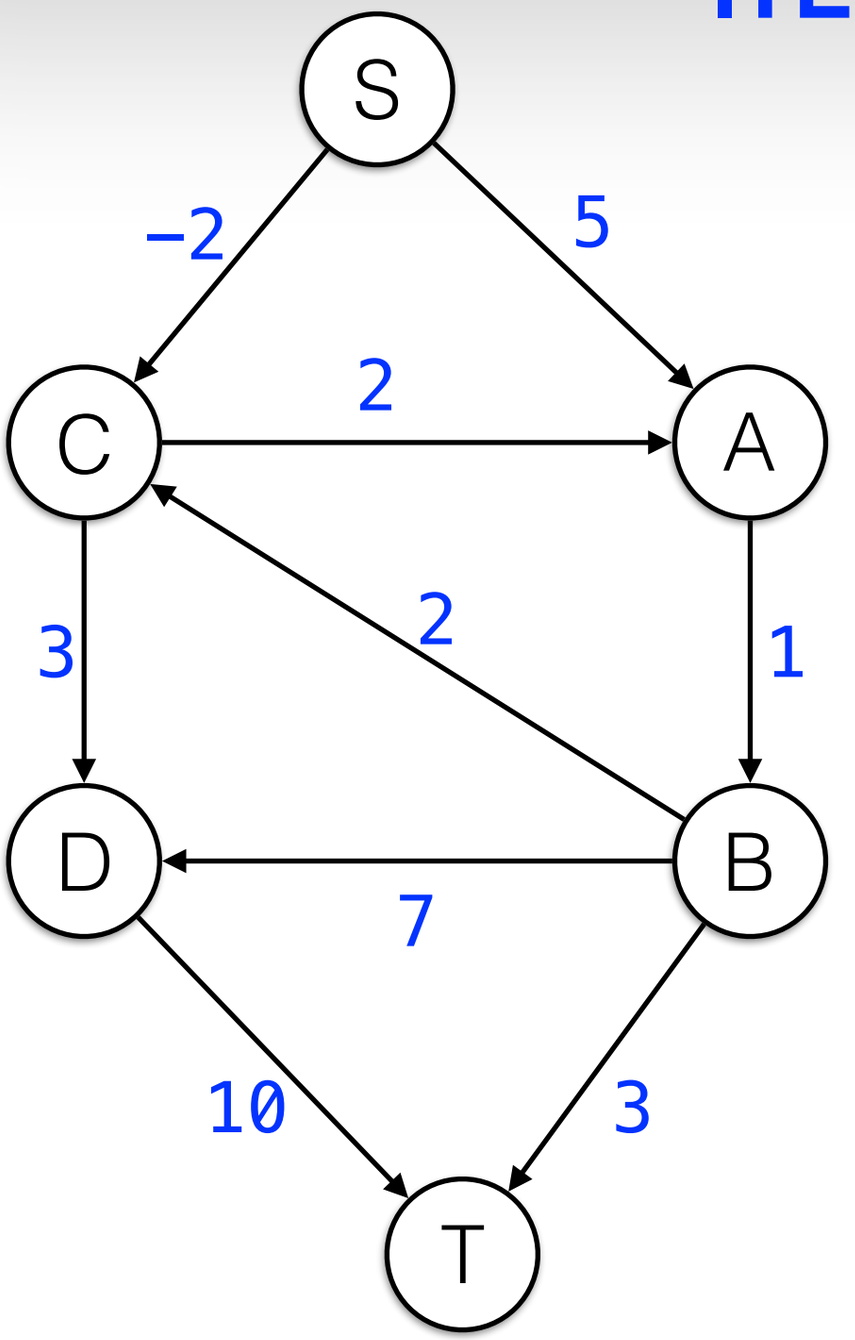
ITERACIÓN 3



v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
S	0	null
A	0	C
B	1	A
C	-2	S
D	1	C
T	4	B

Bellman-Ford

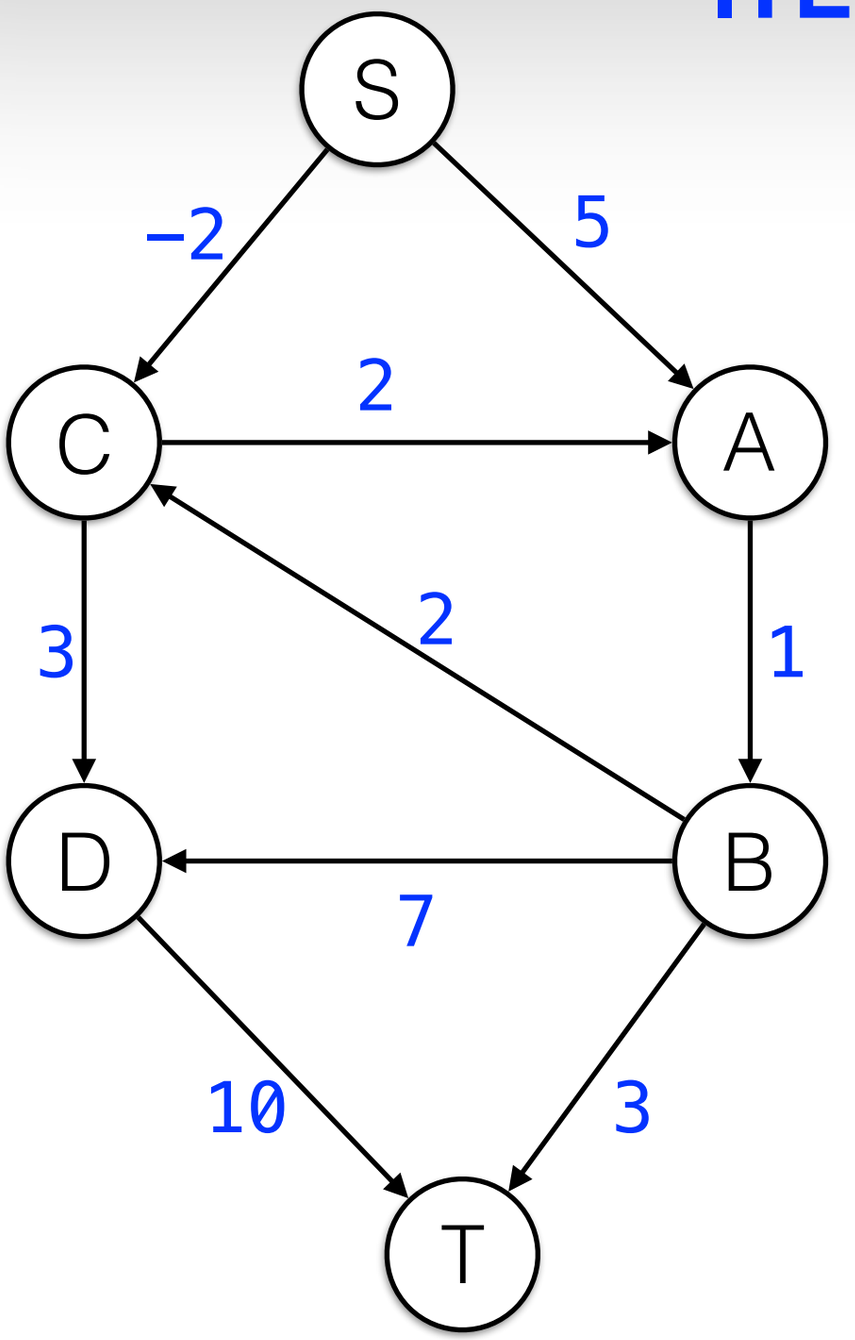
ITERACIÓN 4



v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
S	0	null
A	0	C
B	1	A
C	-2	S
D	1	C
T	4	B

Bellman-Ford

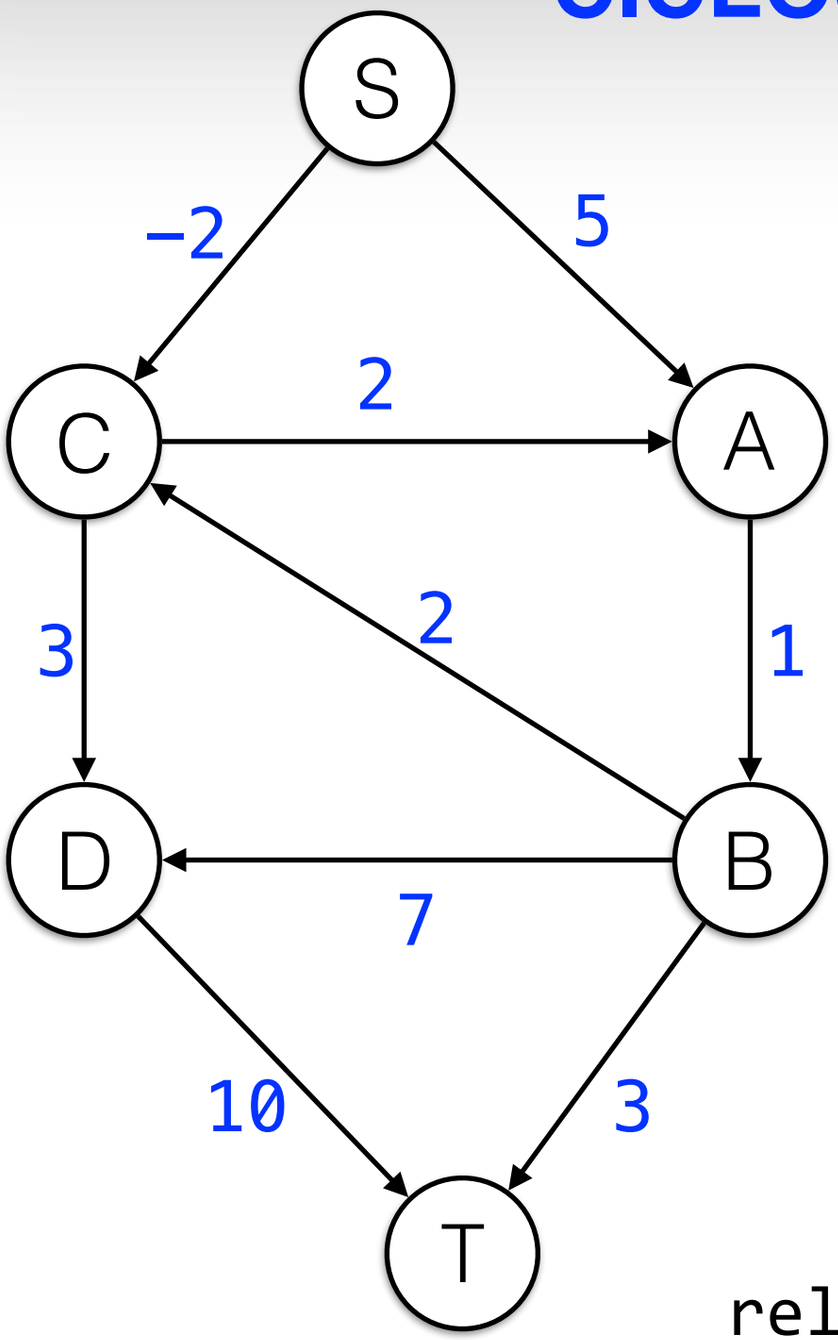
ITERACIÓN 5



v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
S	0	null
A	0	C
B	1	A
C	-2	S
D	1	C
T	4	B

Bellman-Ford

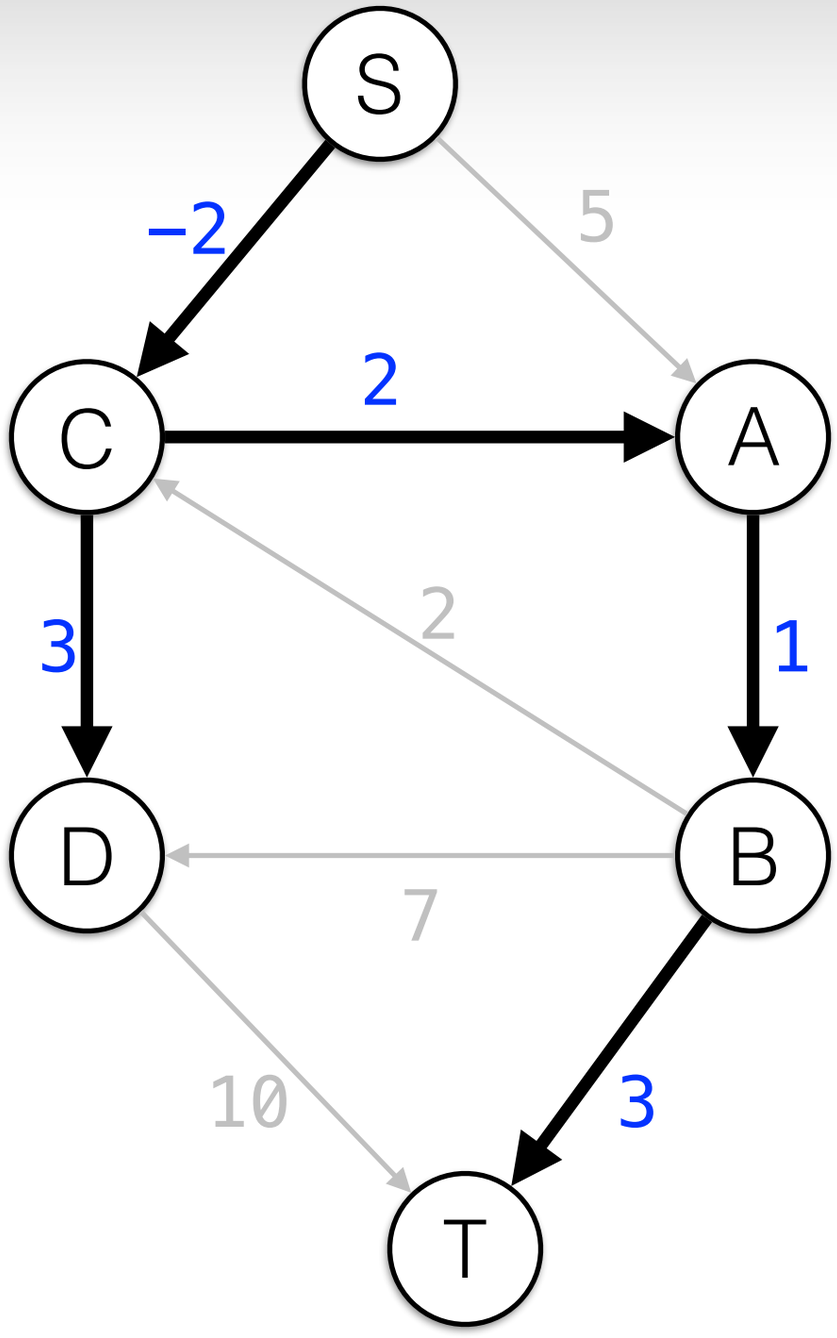
CICLOS NEGATIVOS



v	$\delta(s,v)$ distTo[v]	$\pi(v)$ edge[v]
S	0	null
A	0	C
B	1	A
C	-2	S
D	1	C
T	4	B

```
relax ? true : false
```

Bellman-Ford

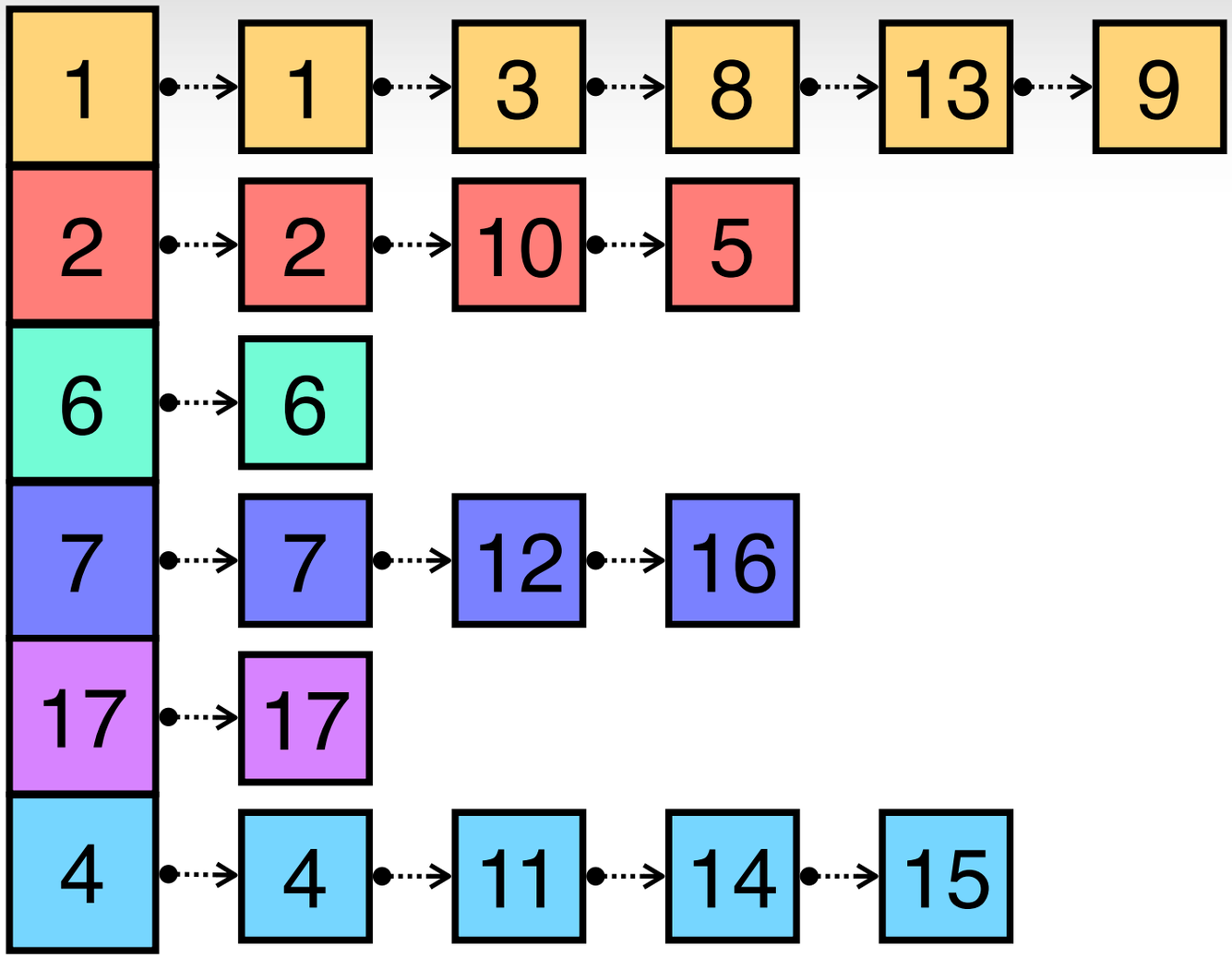


UNION FIND

[Cormen et al. Chaper 21]

Union-find

Disjoint sets



Set size

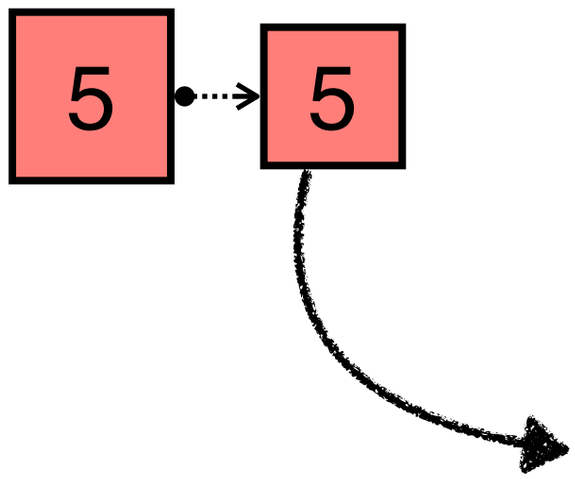
1	5
2	3
6	1
7	3
17	1
4	4

Sets array

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

make

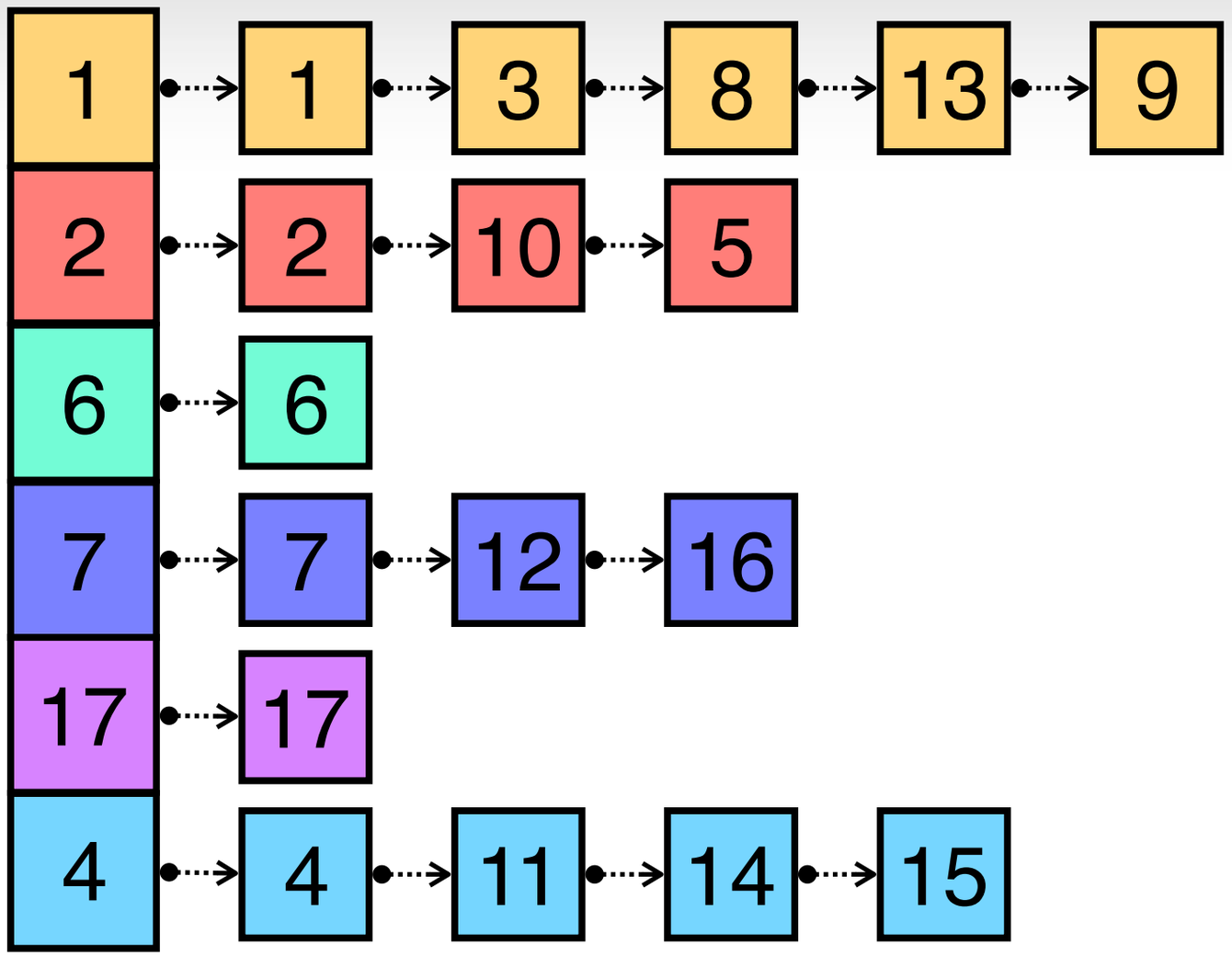
```
make(5)
```



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

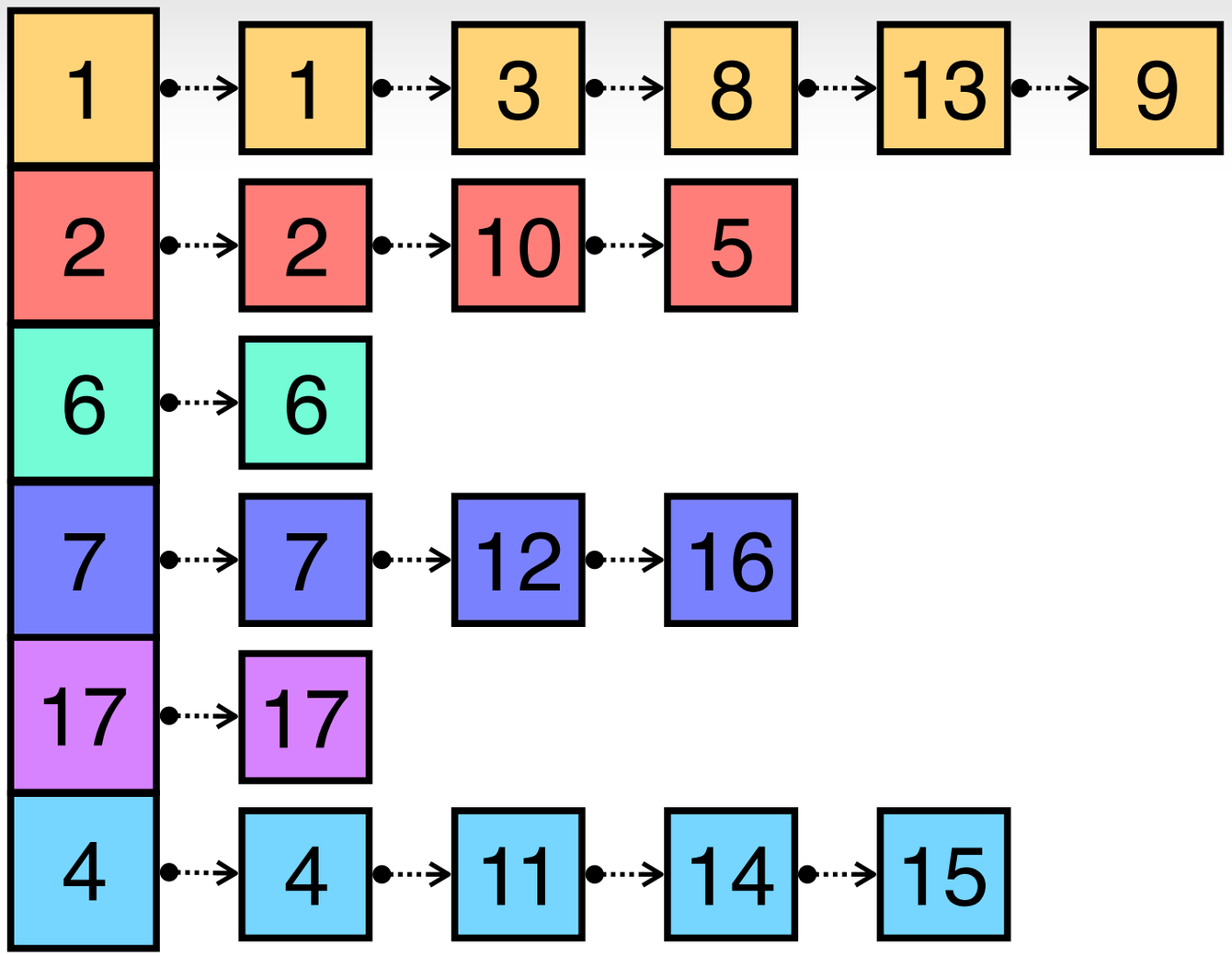
find

`find(5)`



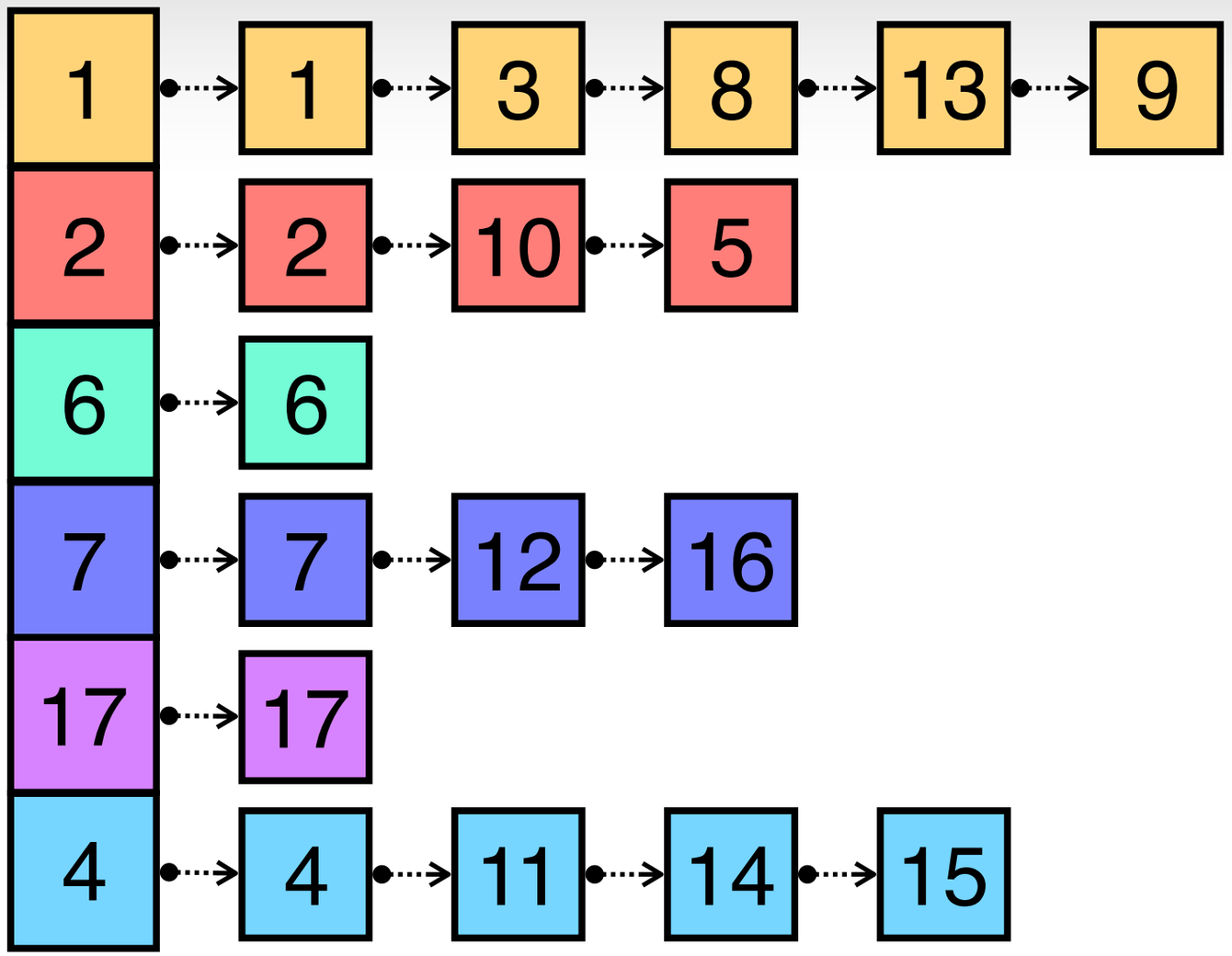
find

```
find(5)
    =
```



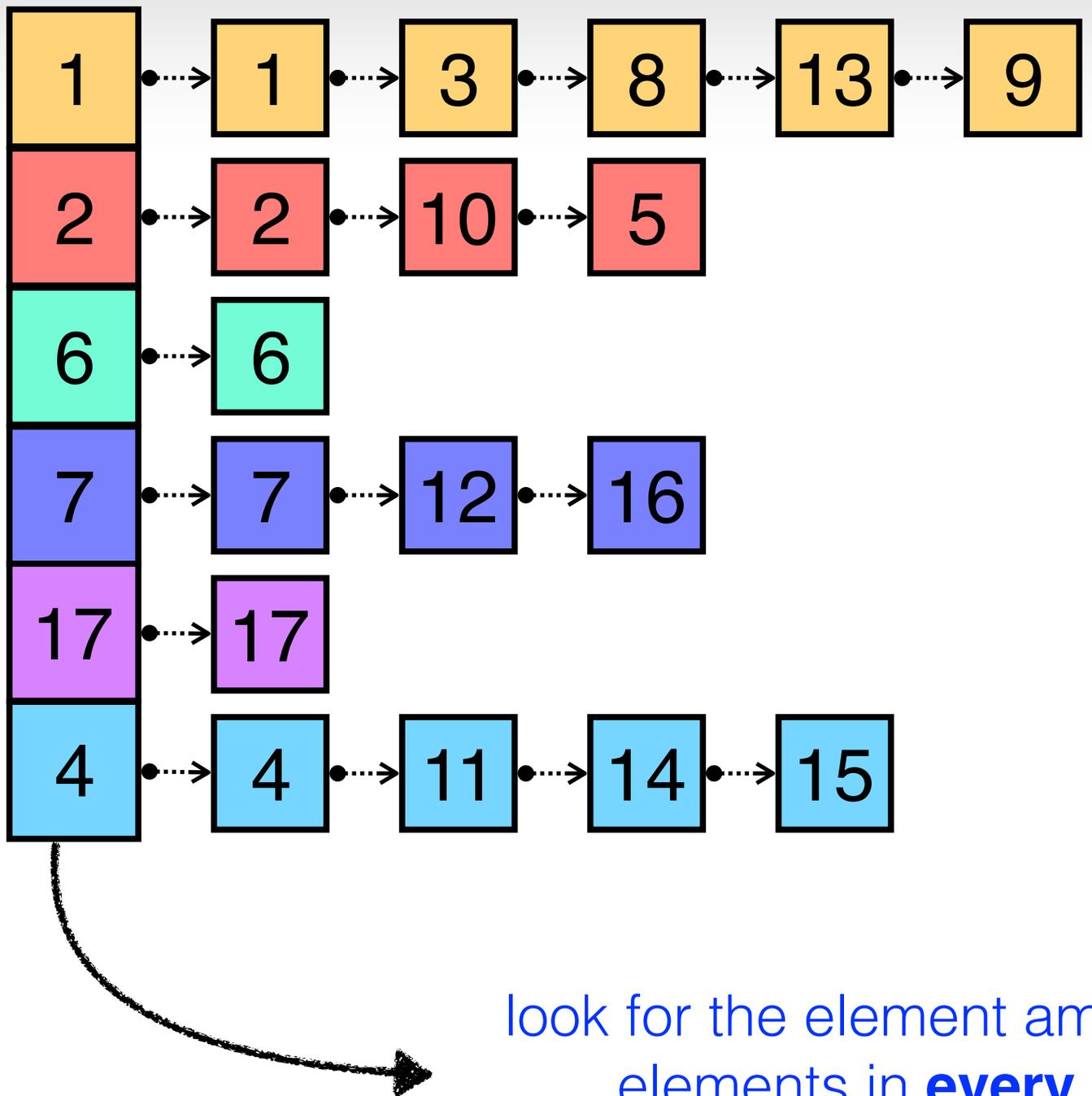
find

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



find

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



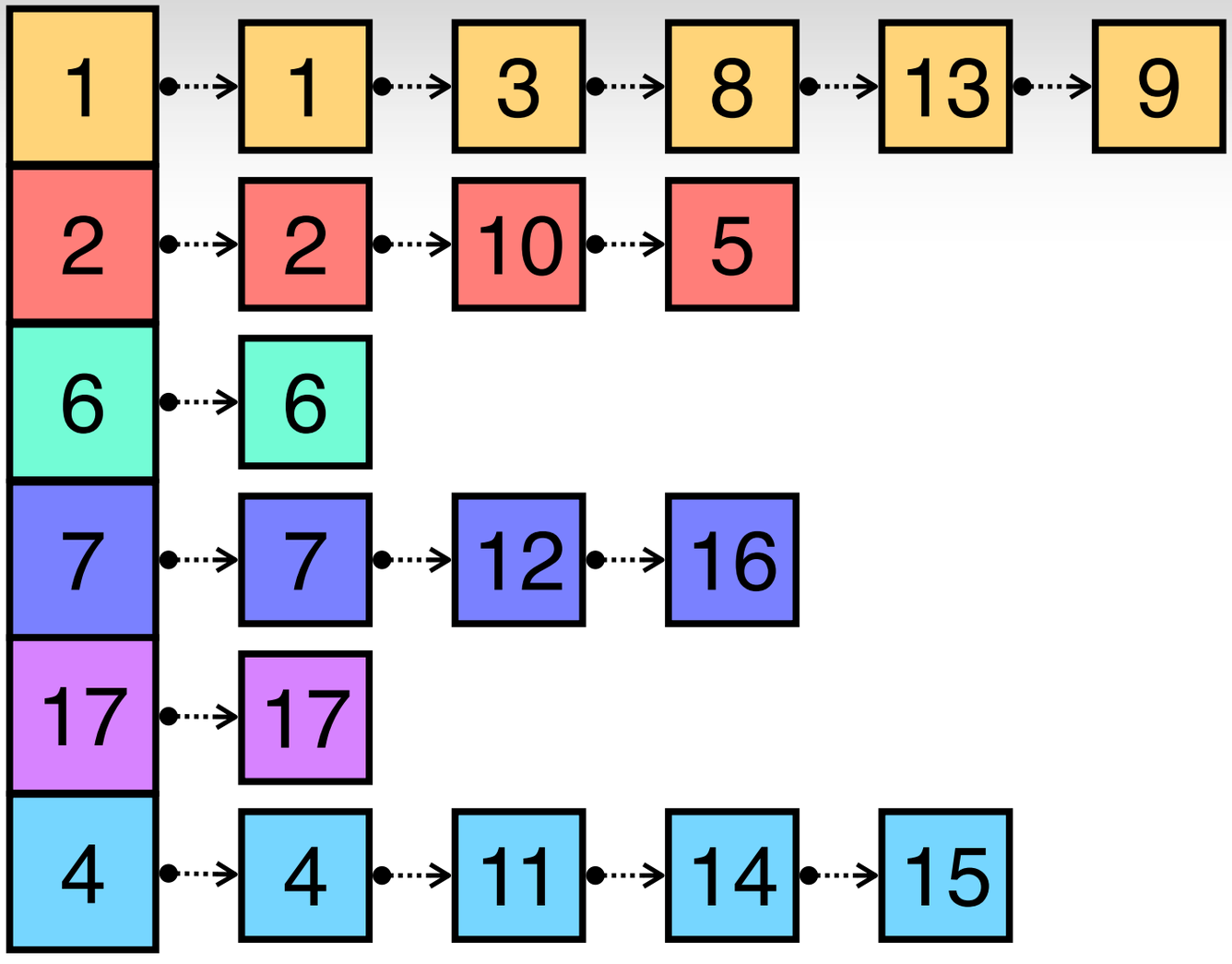
look for the element among **all** elements in **every** set

CUÁL ES LA COMPLEJIDAD DE FIND?

find

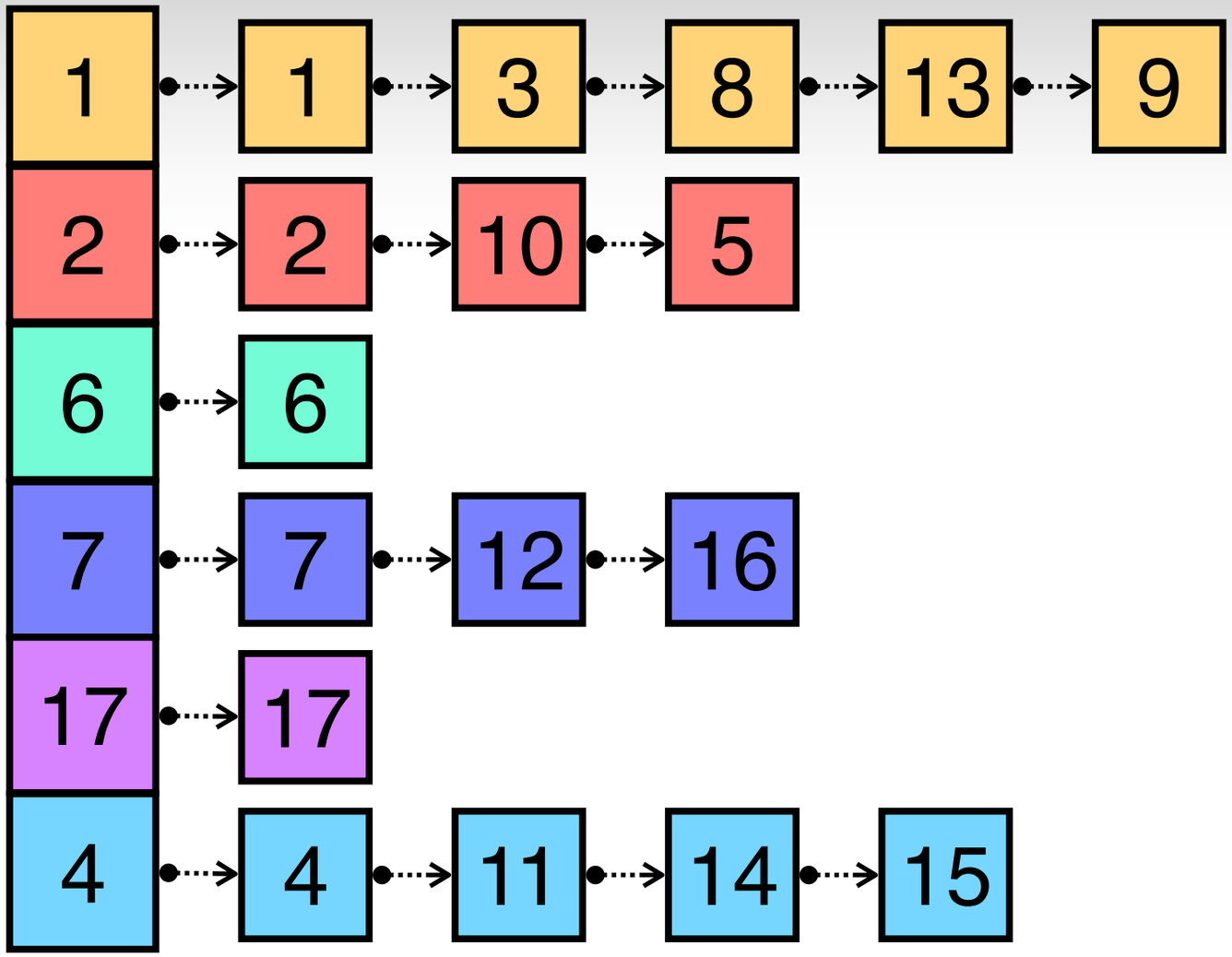
`find(5)`

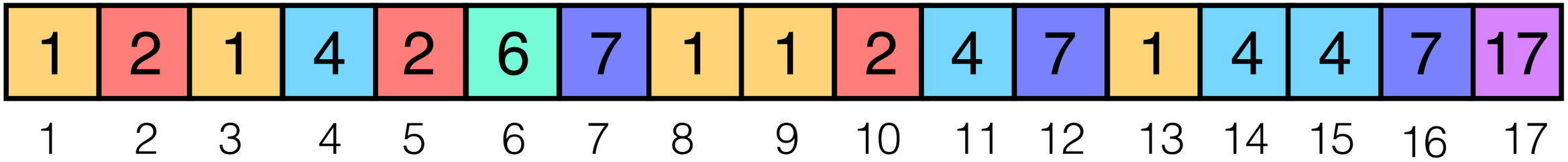
O(N)



find

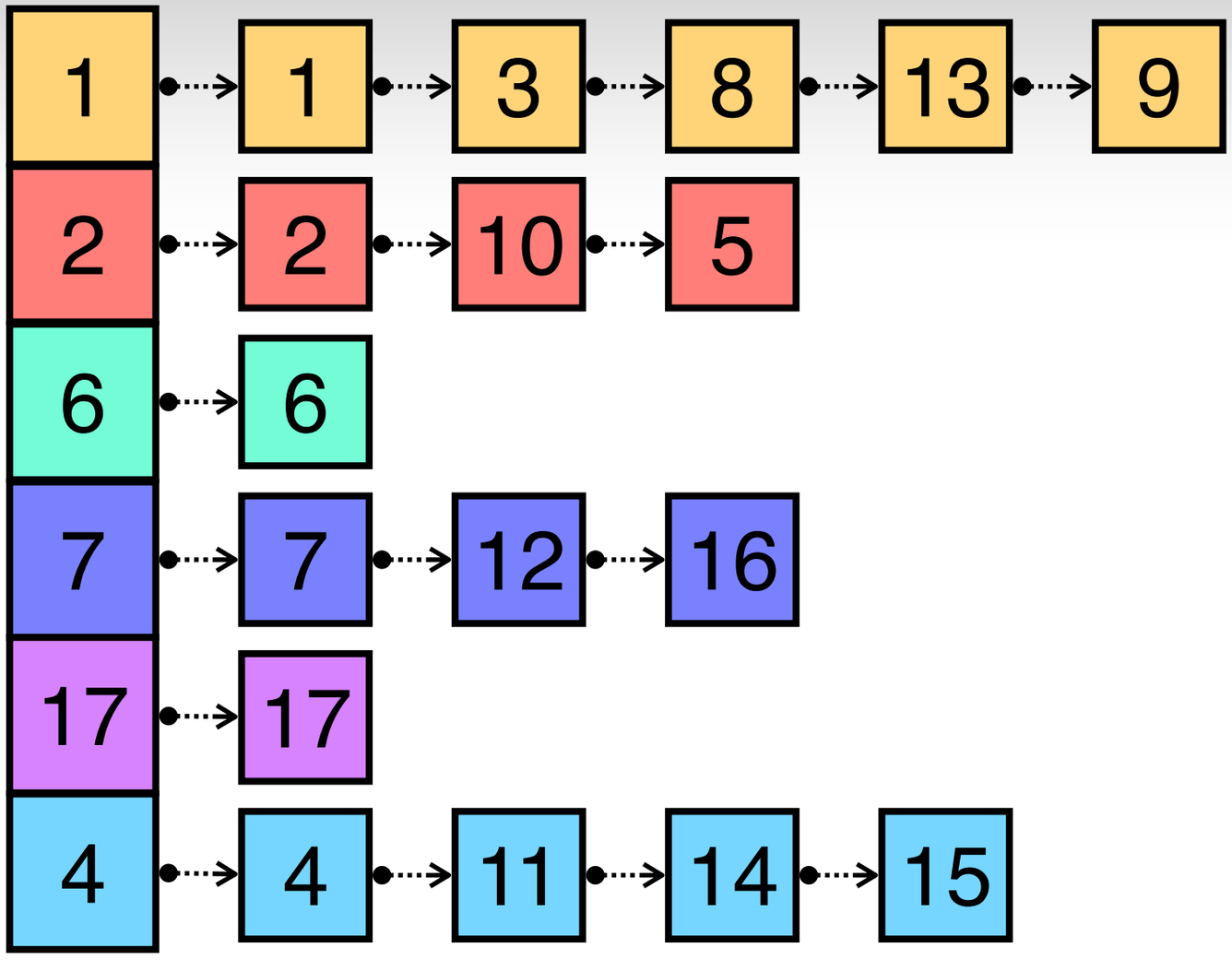
`find(5)`


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

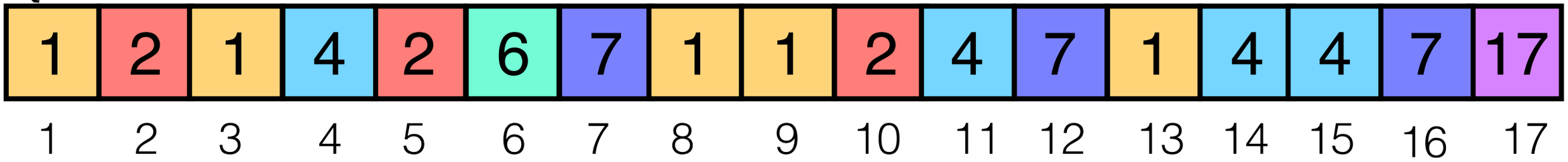

1 2 1 4 2 6 7 1 1 2 4 7 1 4 4 7 17
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

find

`find(5)`



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

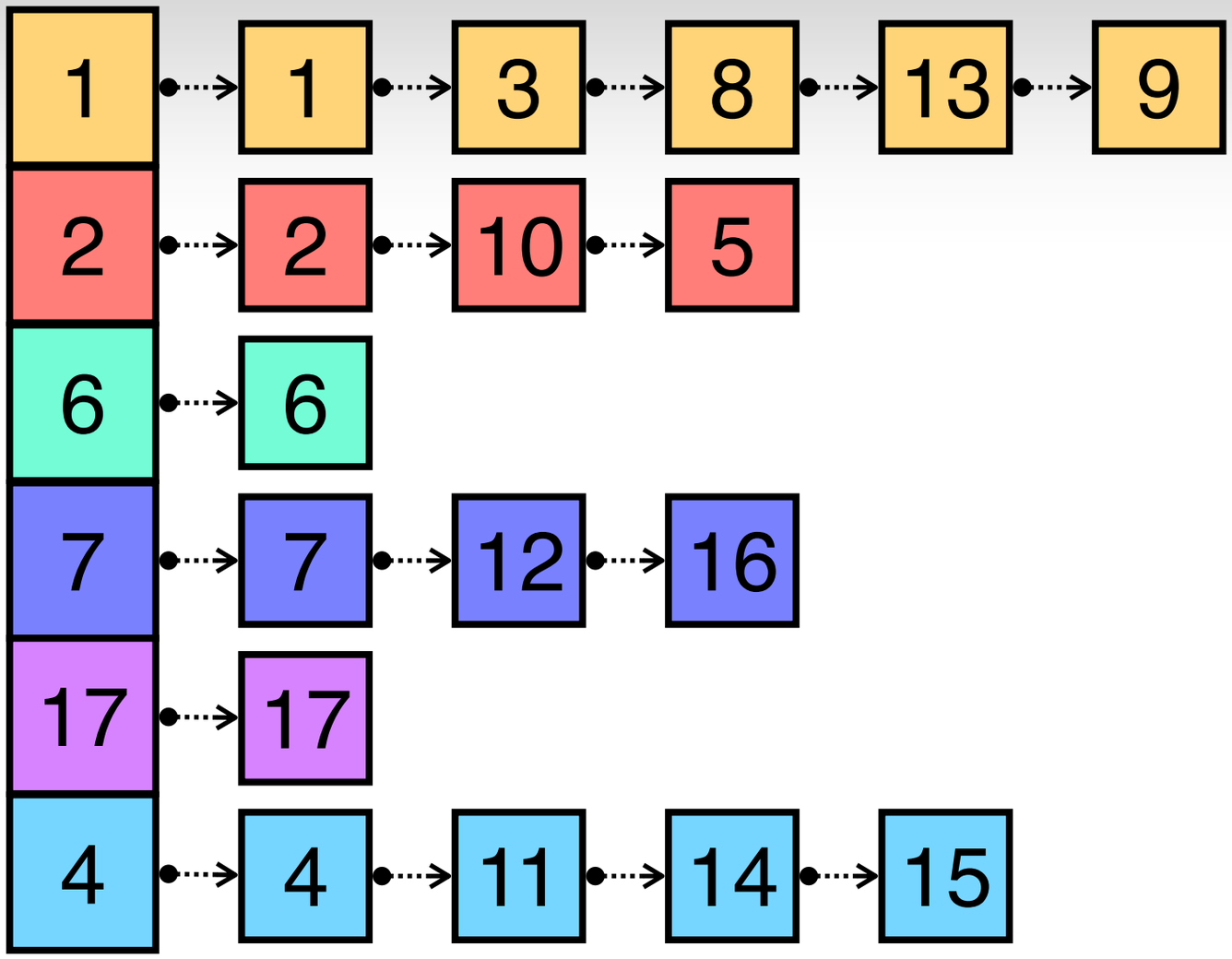


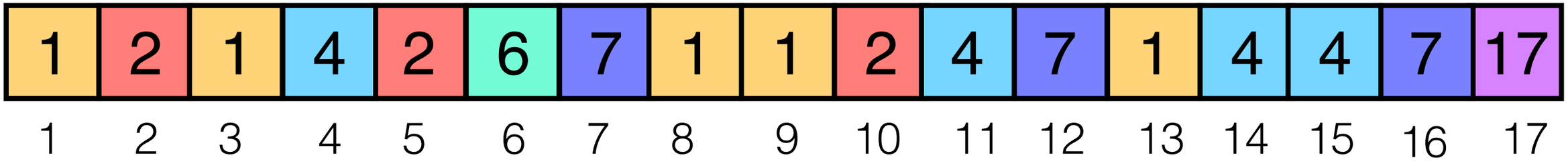
COMPLEXITY?

find

`find(5)`

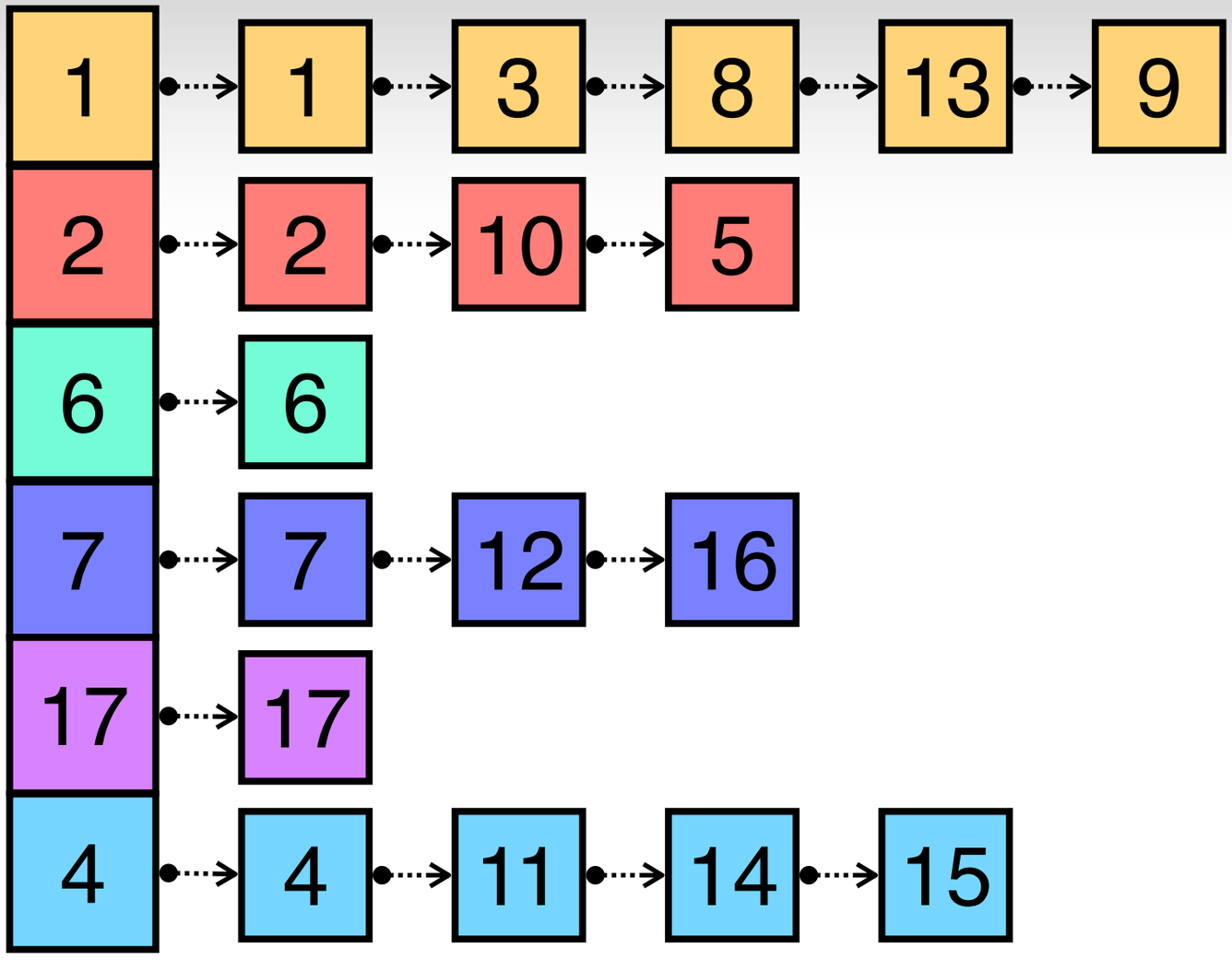
O(1)





count

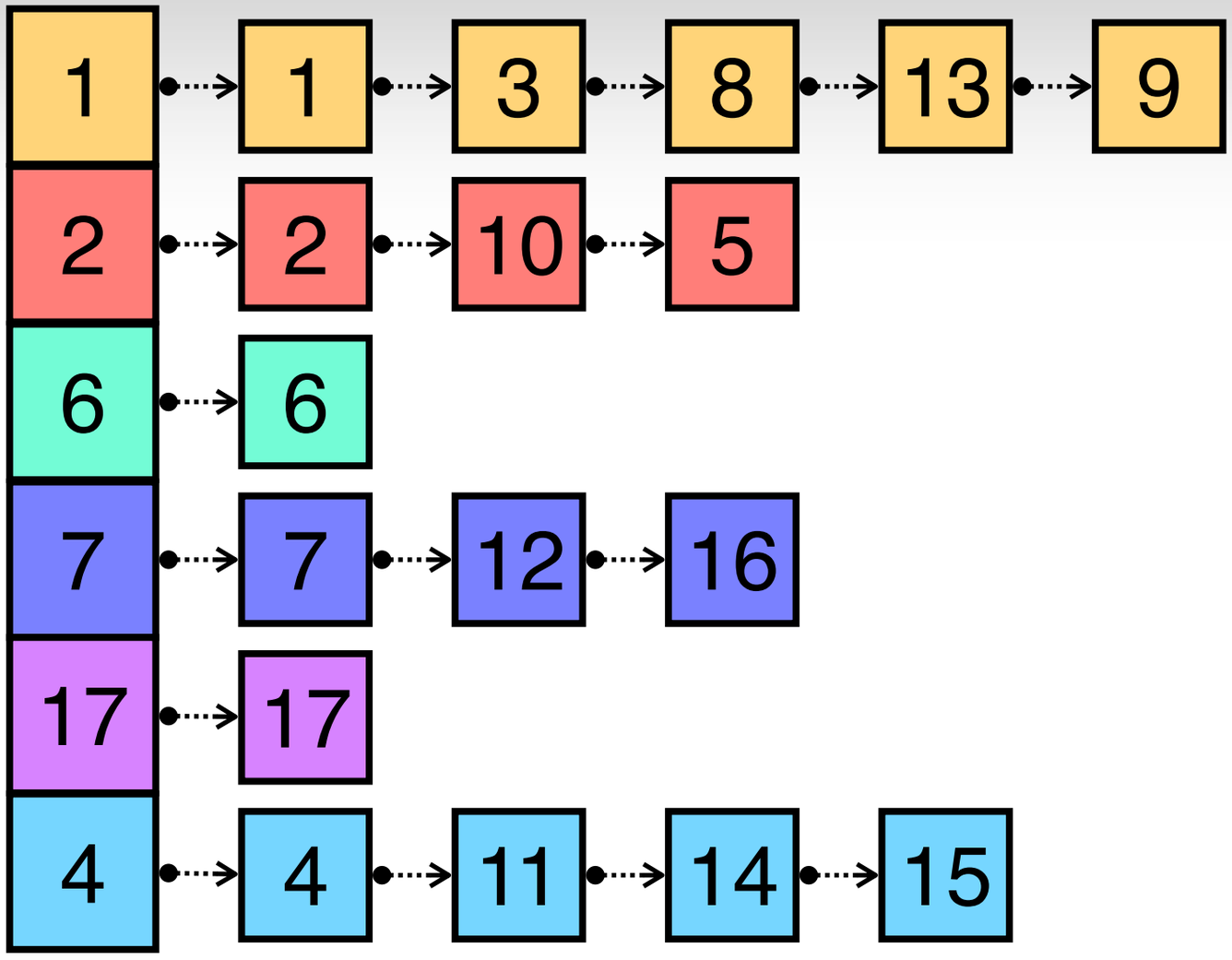
`count(13)`



count

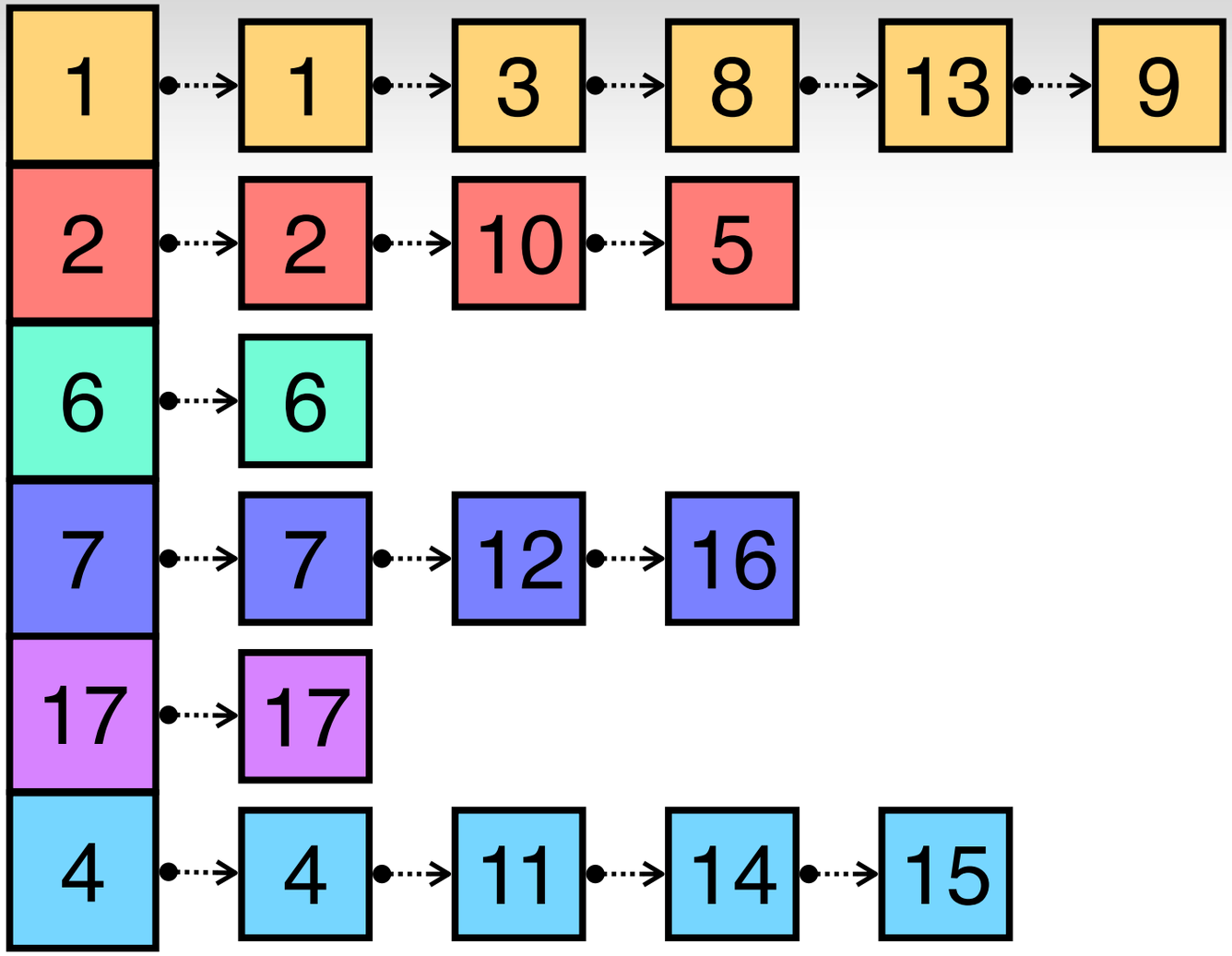
`count(13)`

=



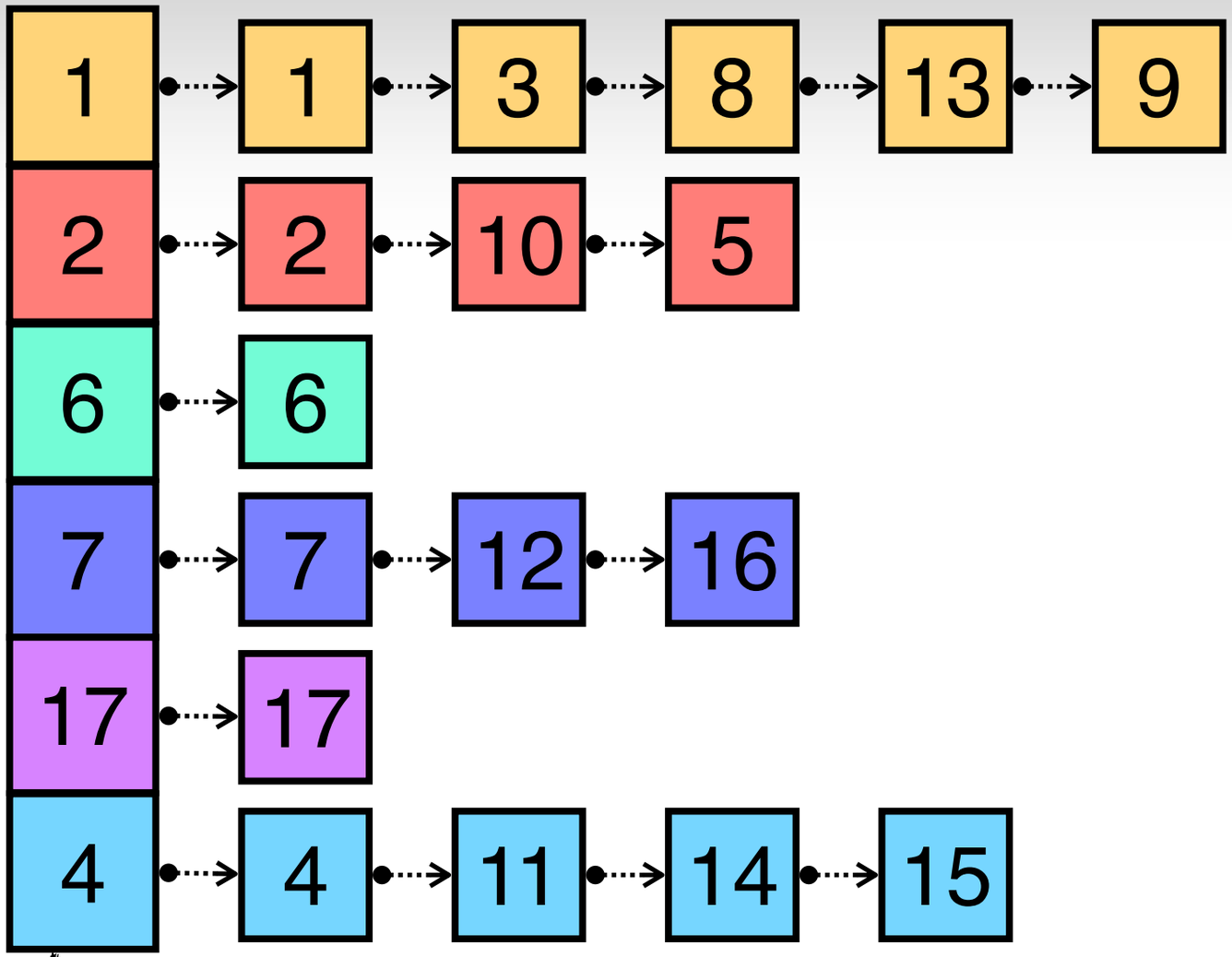
count

```
count(13)
    =
5
```



count

```
count(13)
    =
5
```

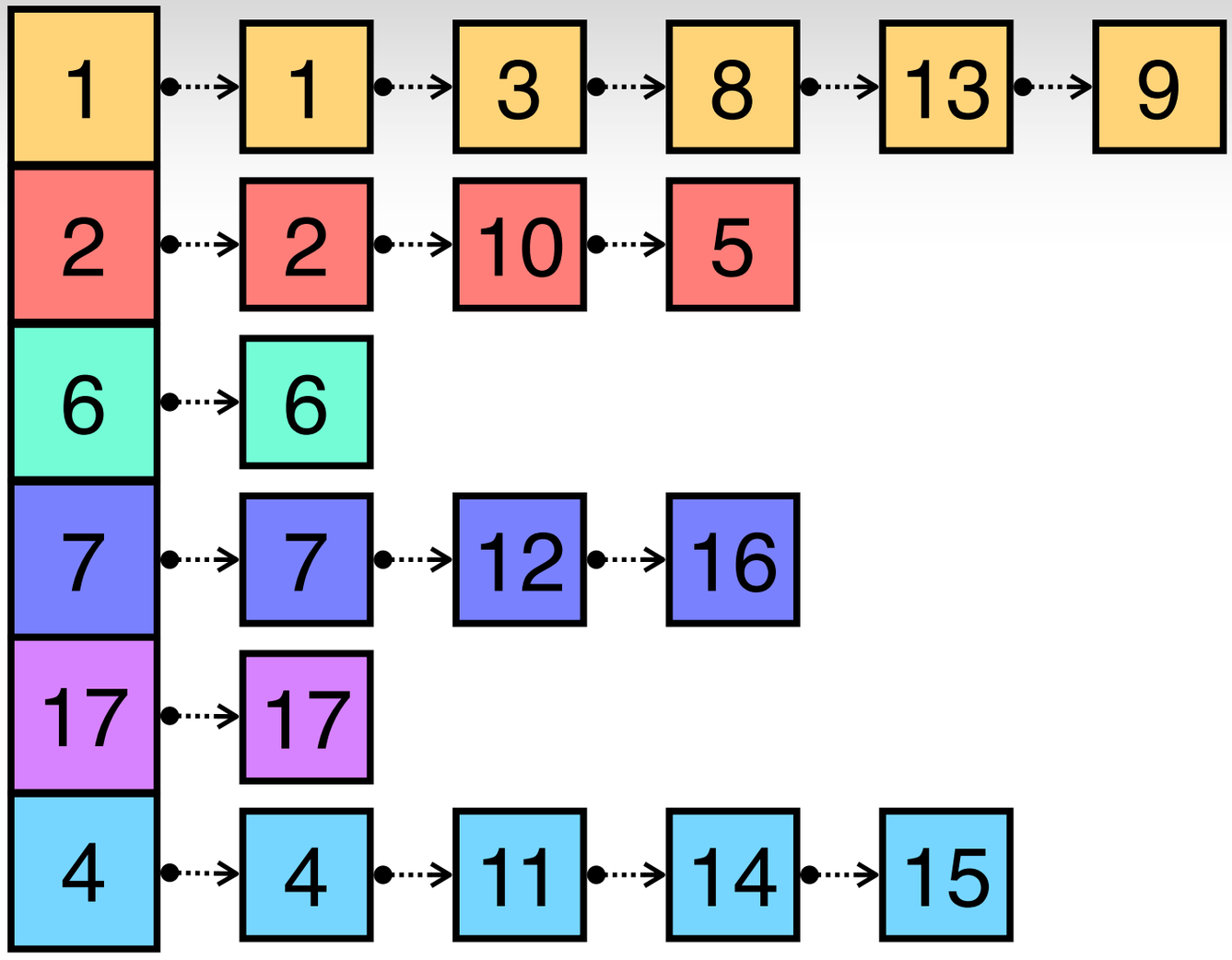


Must search the element **between all** the elements of **every representative** and count the elements of such representative

COMPLEXITY OF COUNT

count

`count(13)`

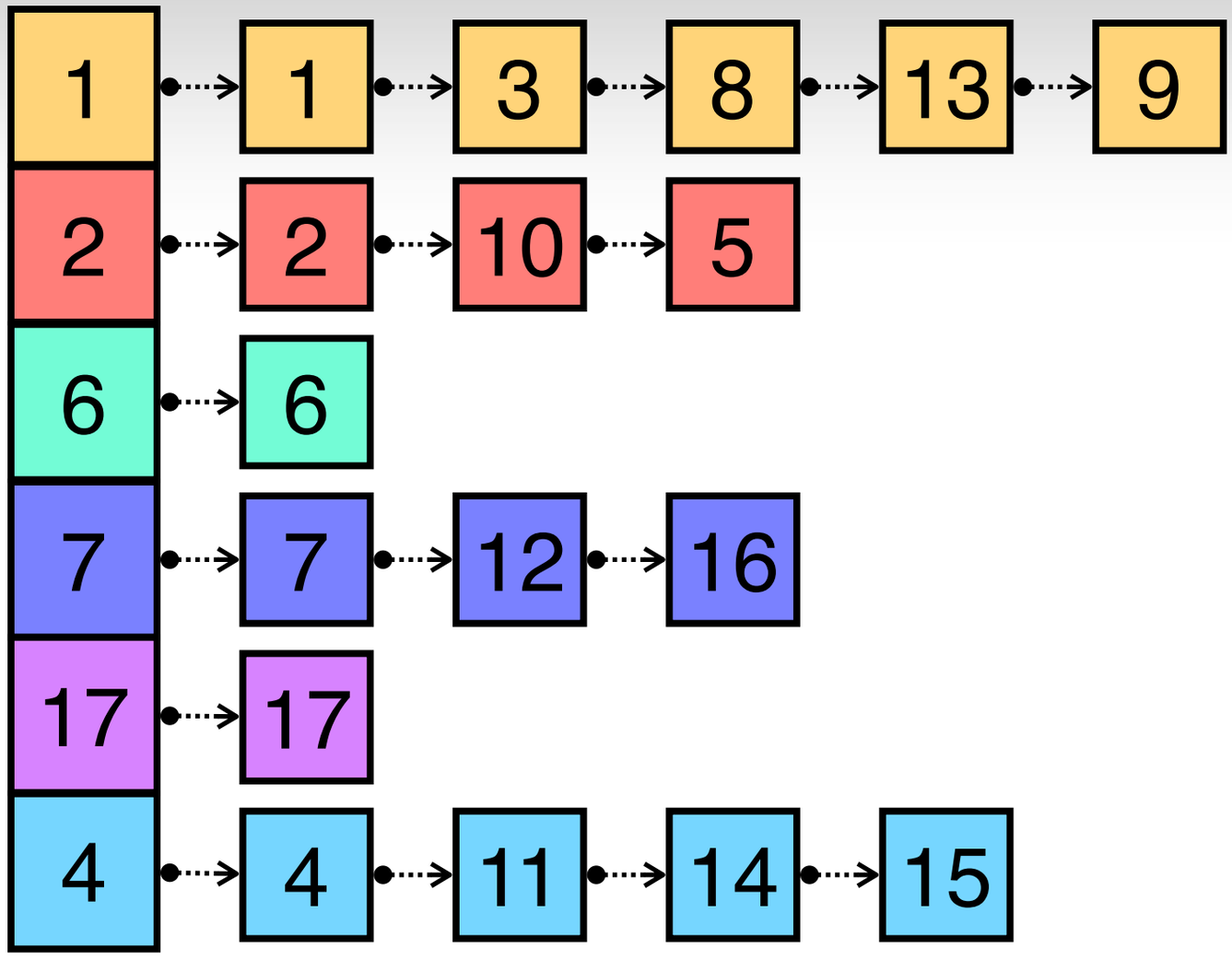


1	5
2	3
6	1
7	3
17	1
4	4

count

`count(13)`

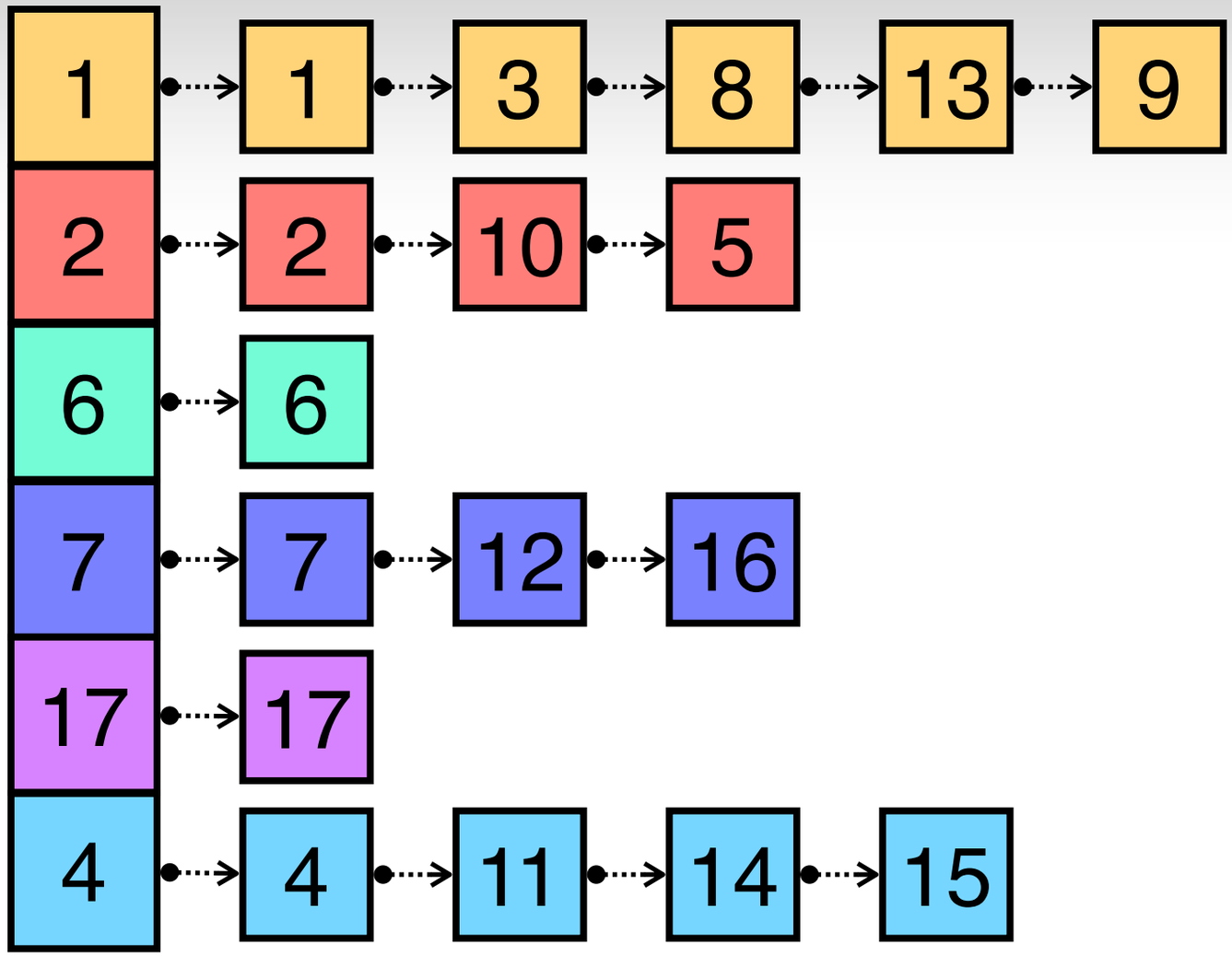
=



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;
  }
}
```

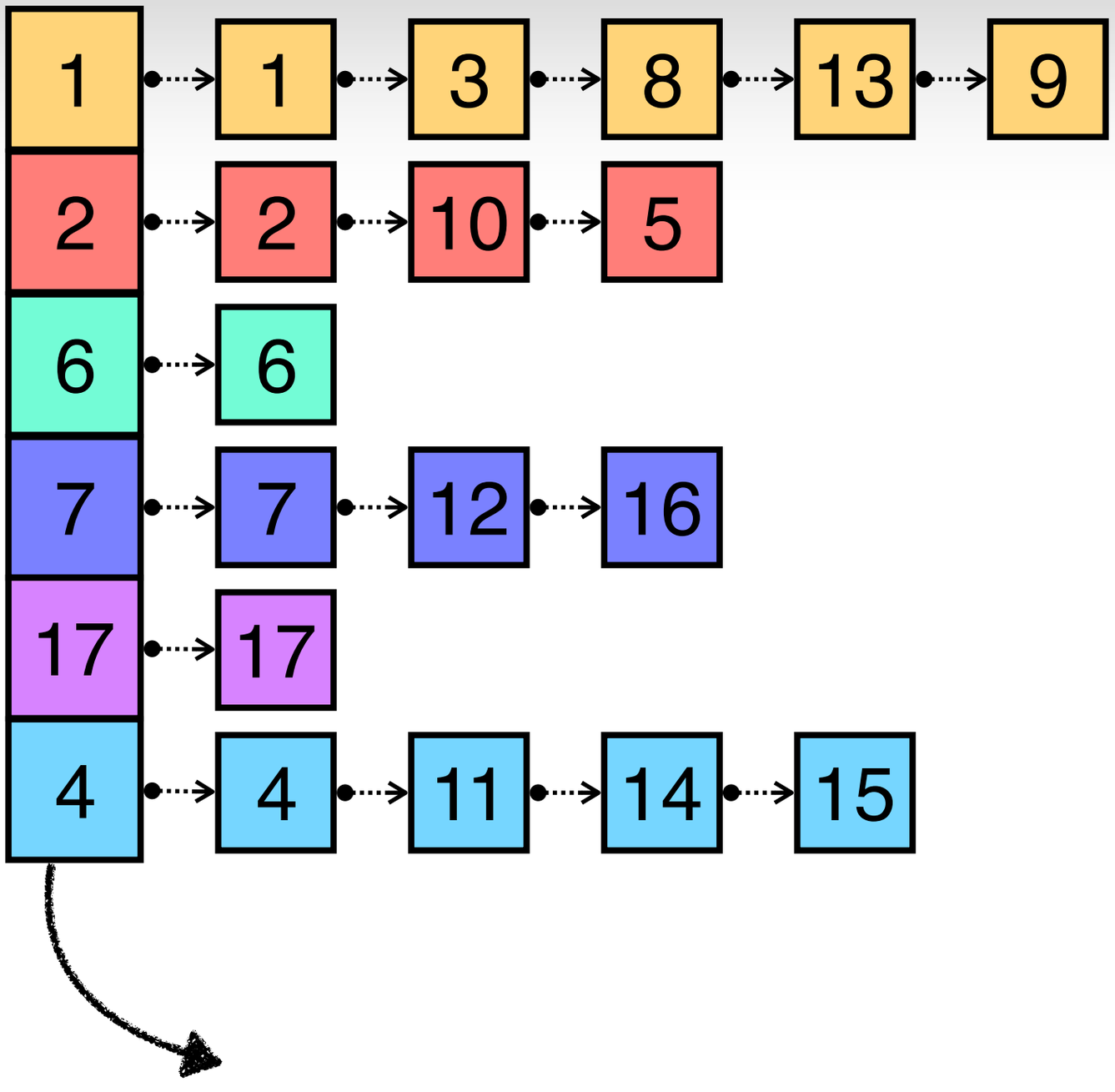
count

```
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|)

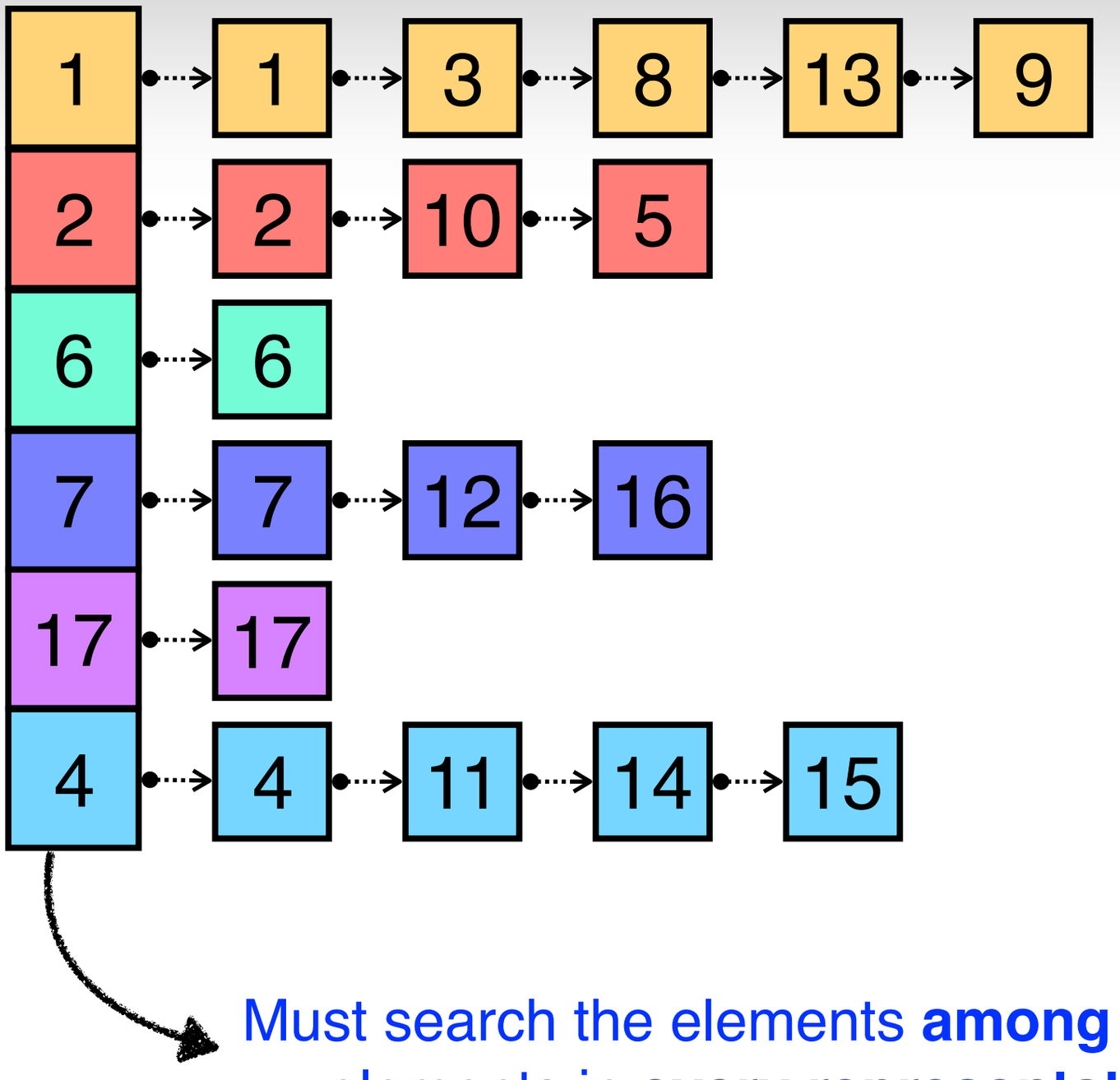
connected

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



connected

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

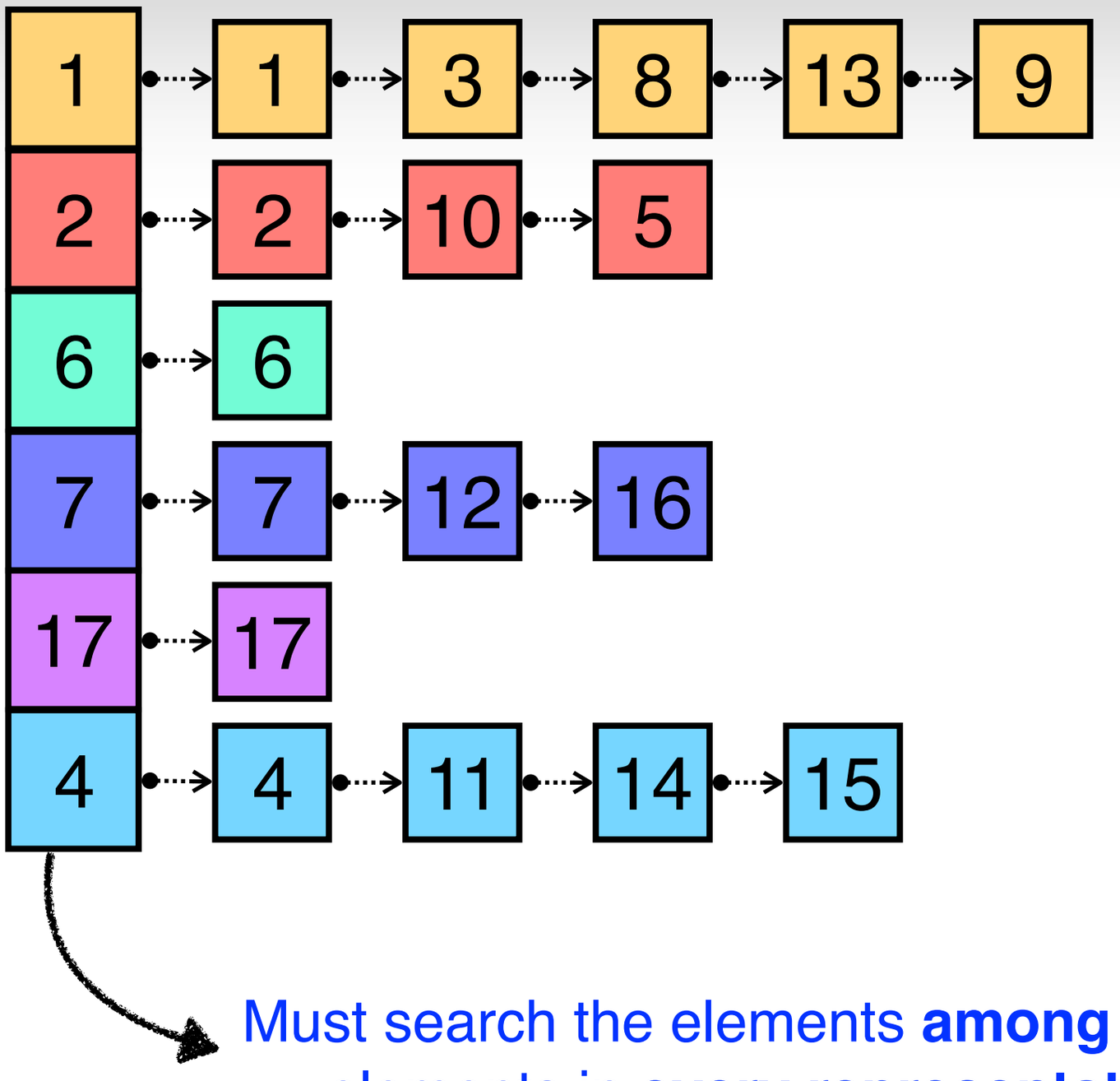


Must search the elements **among all** the elements in **every representative**

connected

`connected(13, 3)`

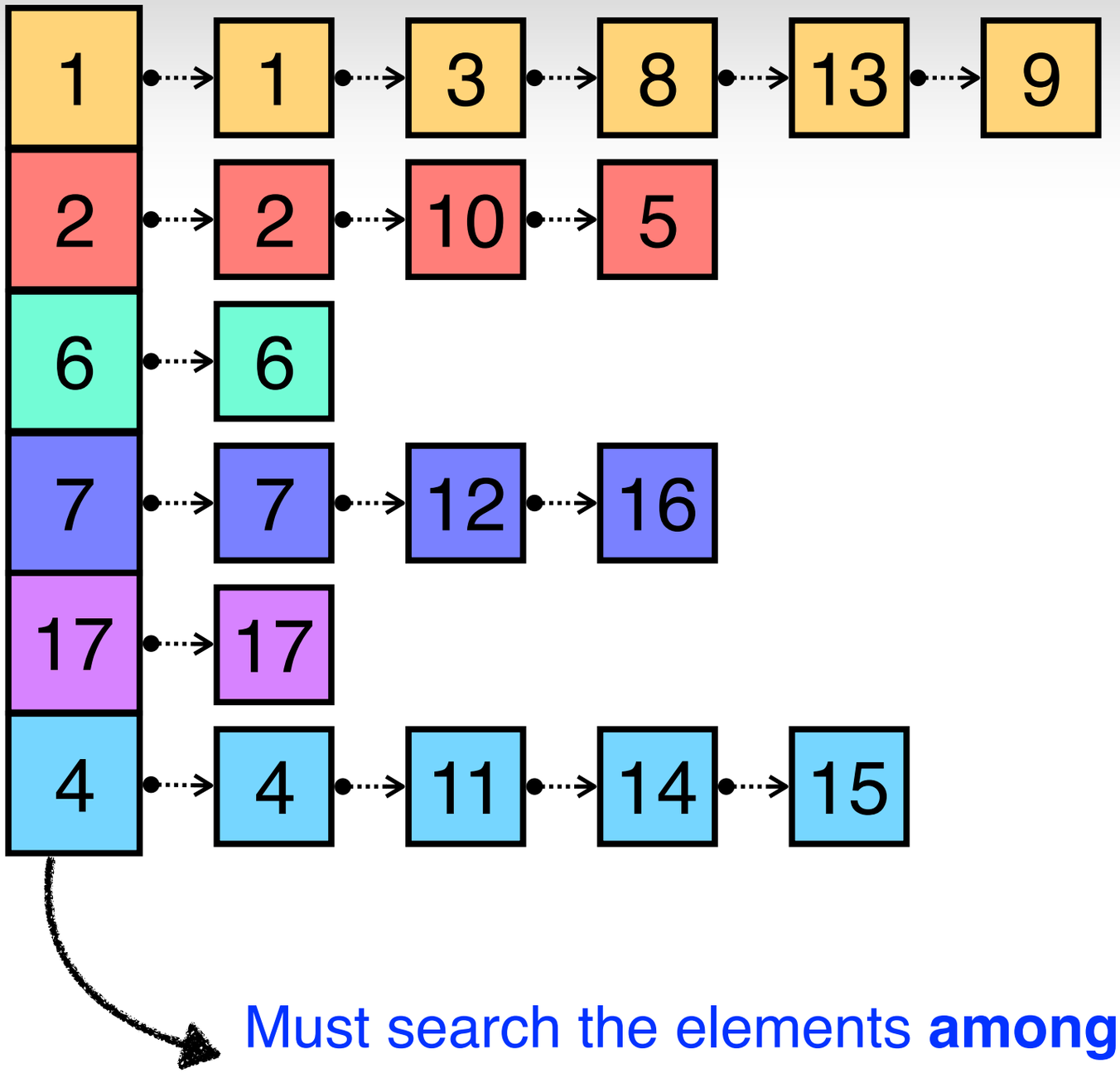
=



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

connected

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



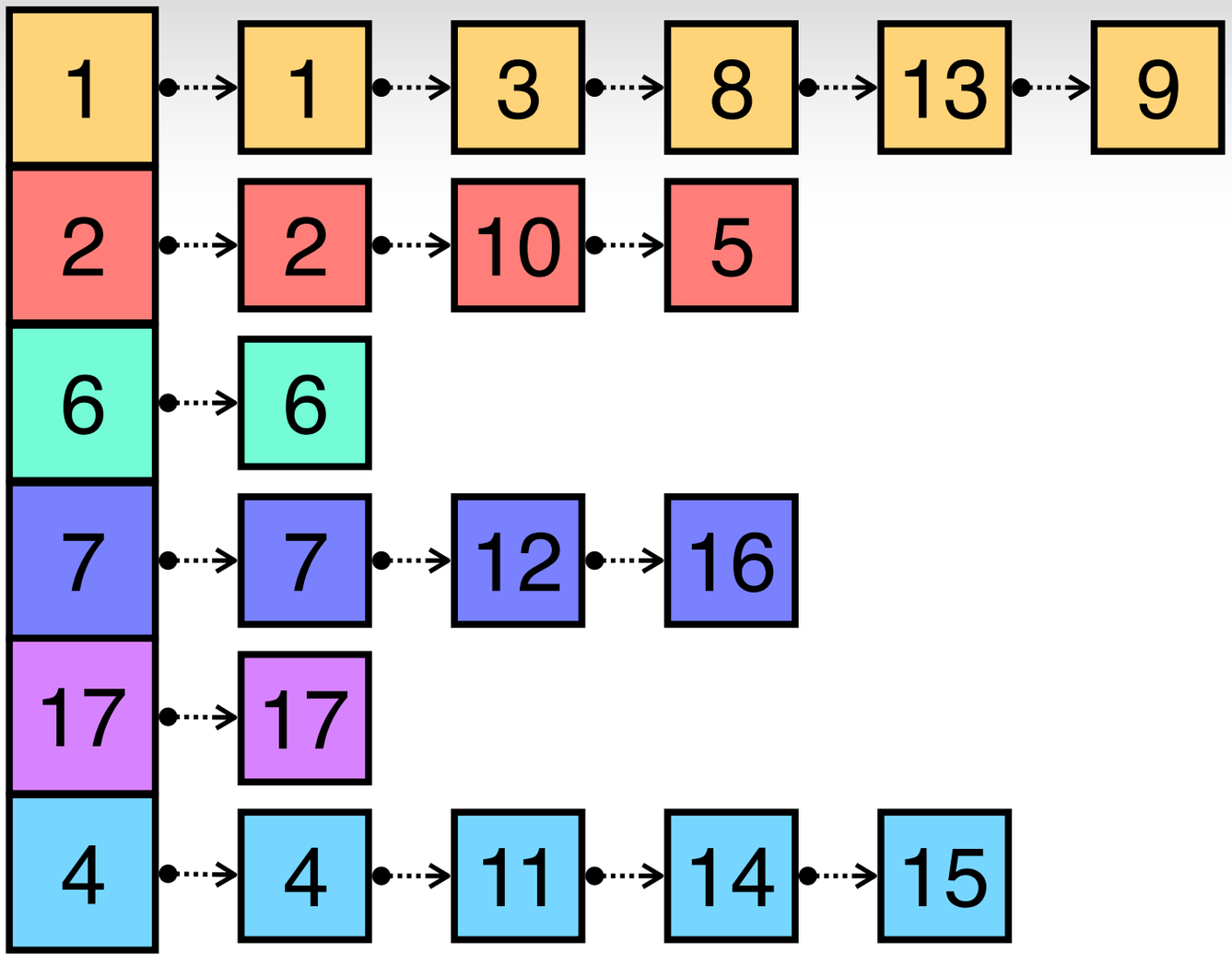
Must search the elements **among all** the elements in **every representative**

COMPLEXITY OF CONNECTED

connected

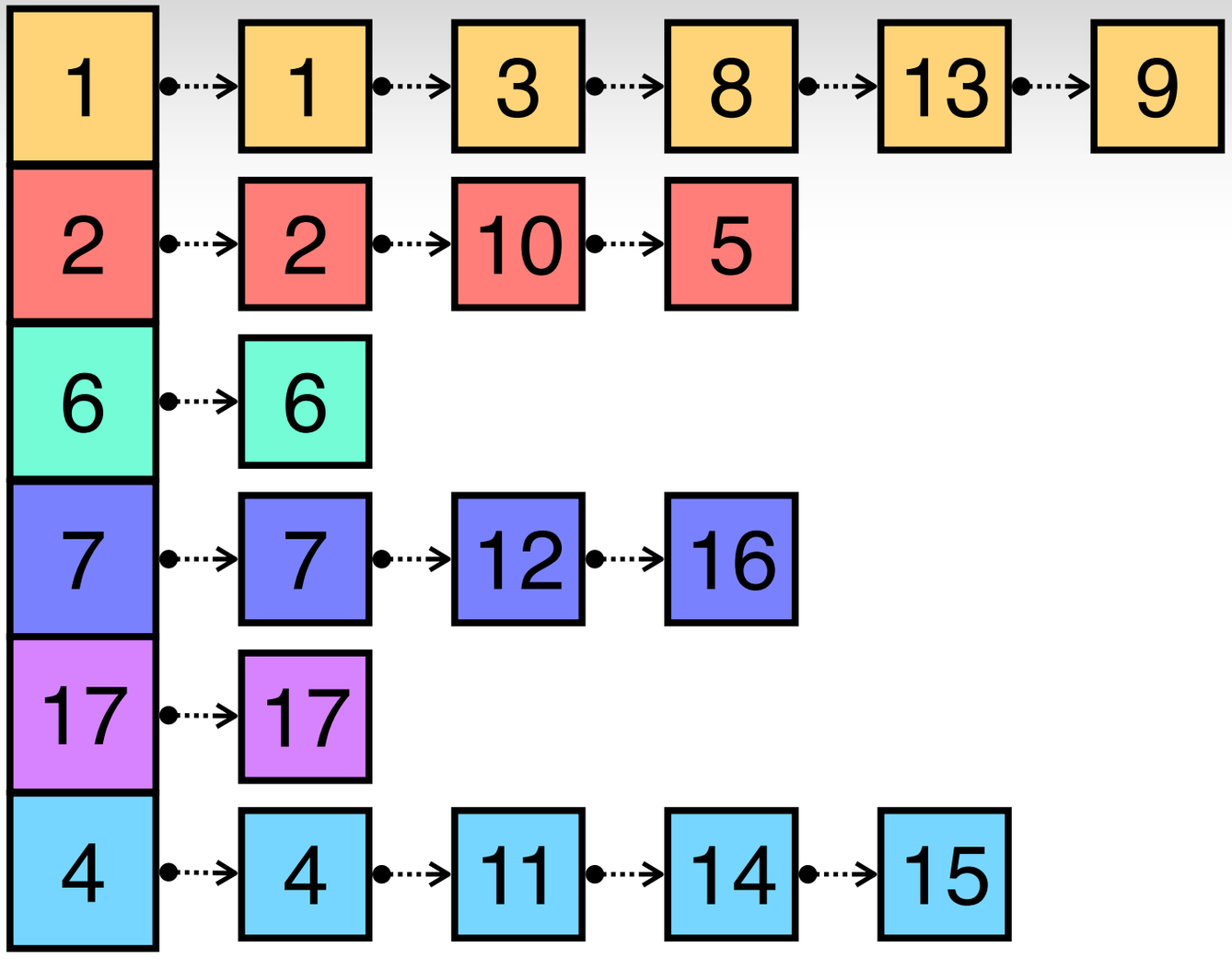
```
connected(13, 3)
```

O(N)



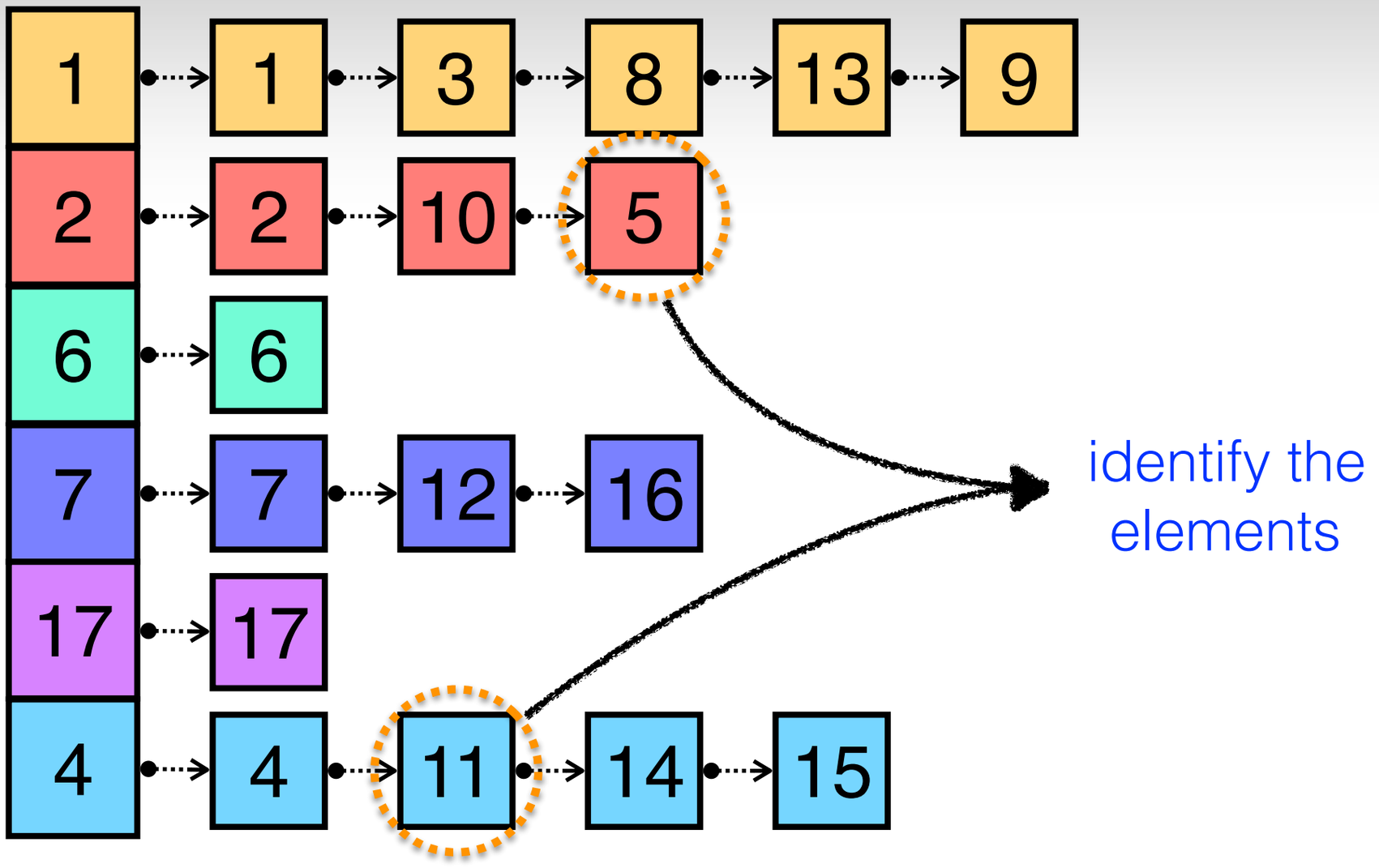
union

`union(5, 11)`



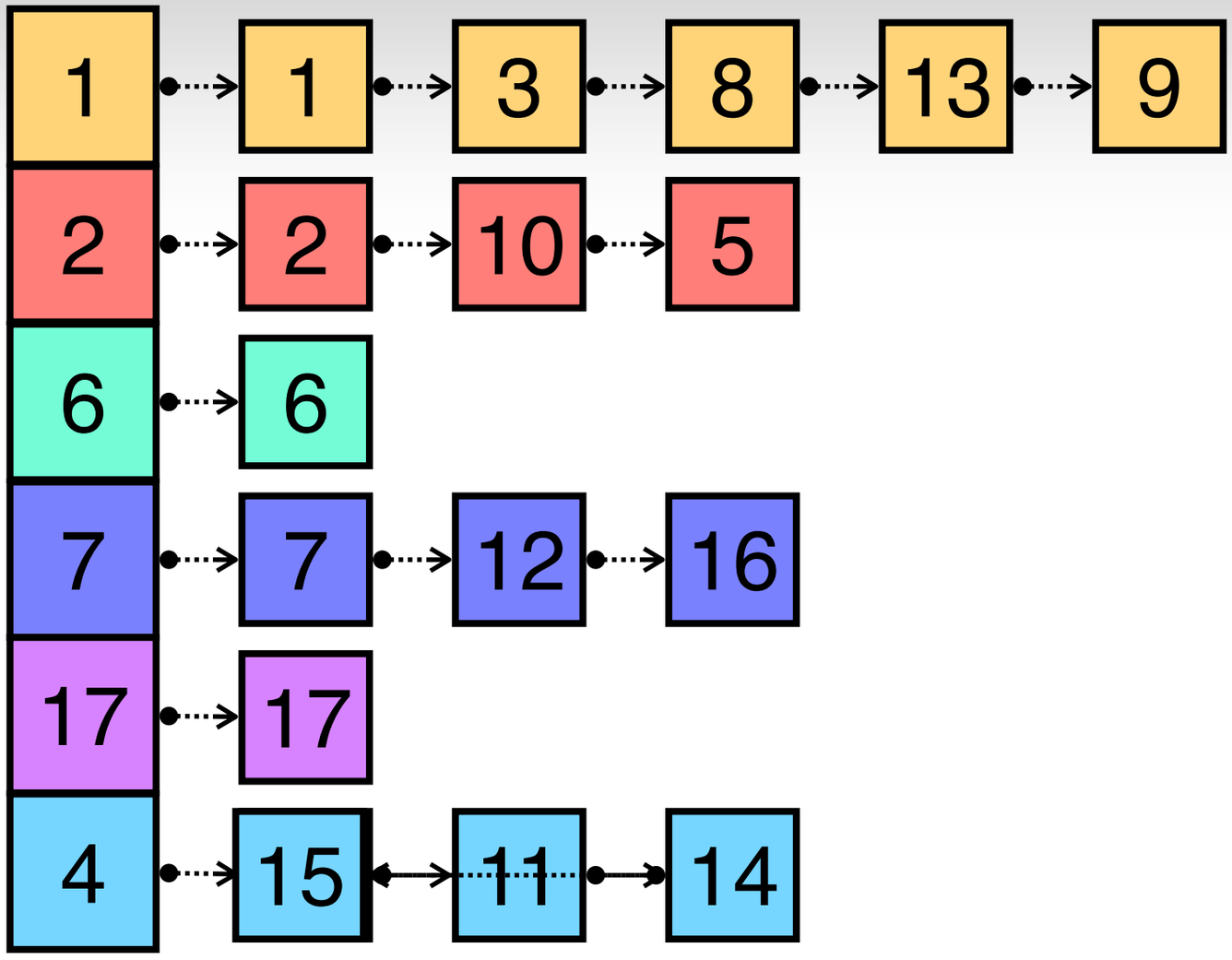
union

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



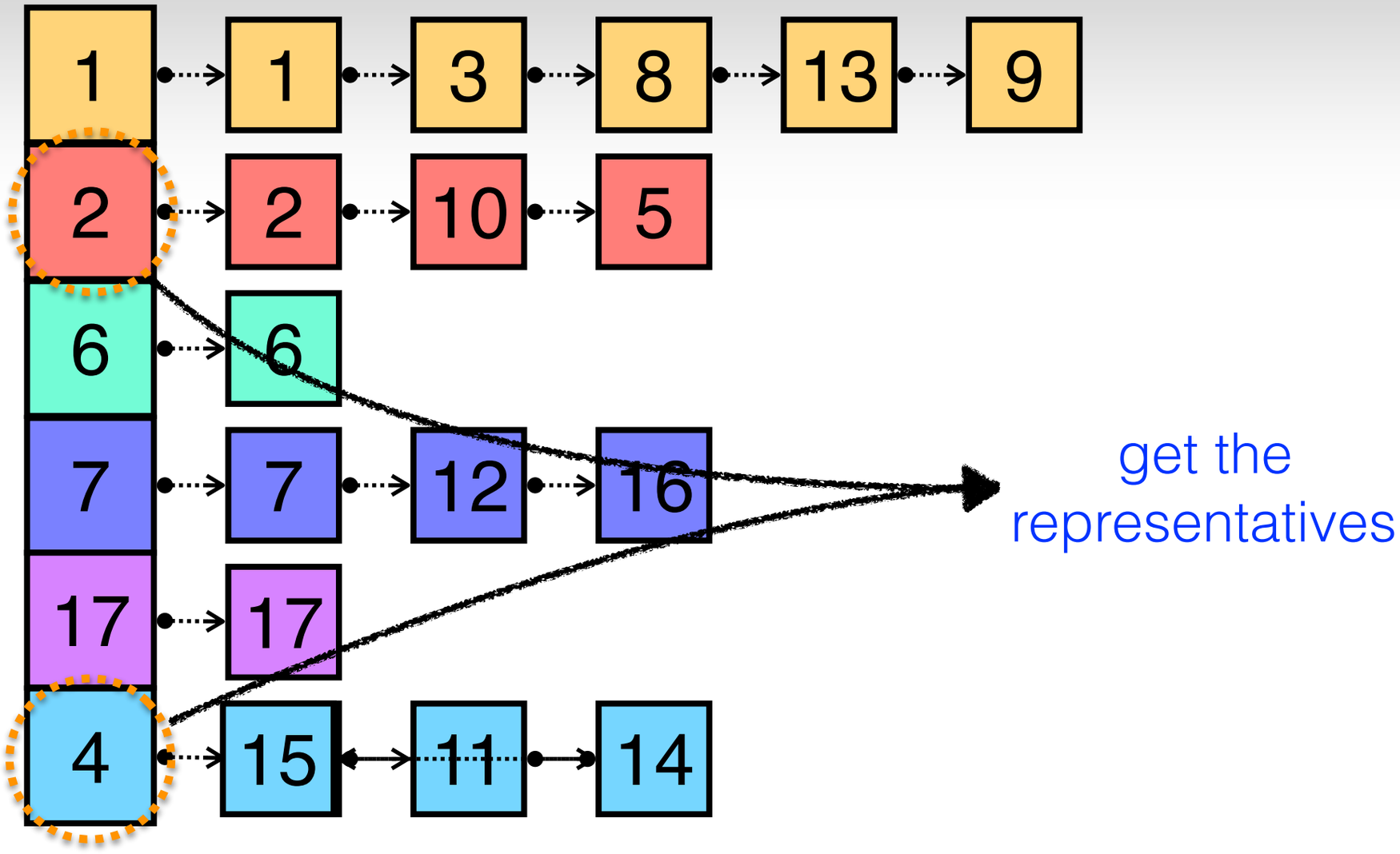
union

`union(5, 11)`



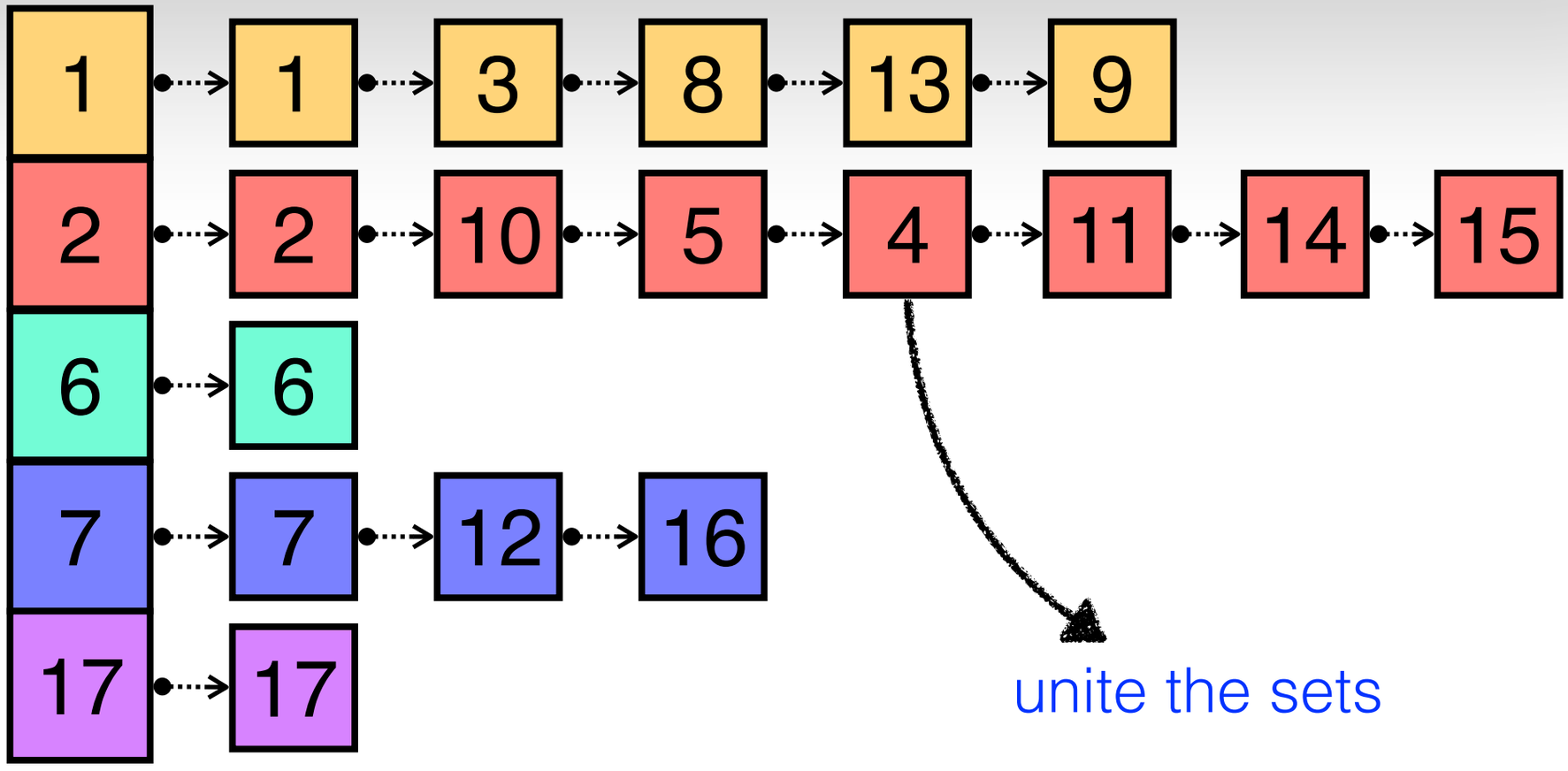
union

`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.
    get(rep1).value);
  else
    sets.get(rep1).value.concat(sets.
    get(rep2).value);
}
```

Union

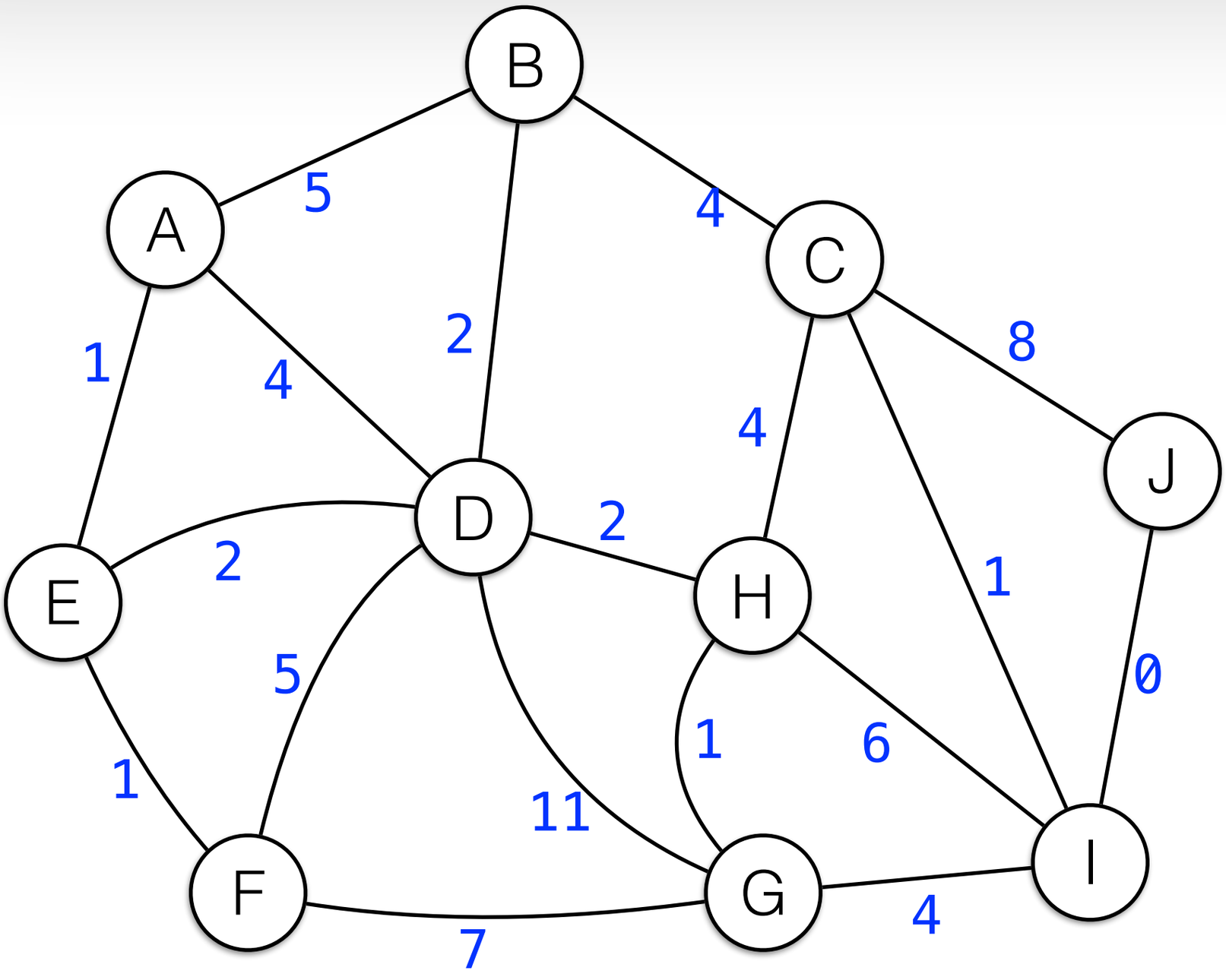
If you do **k** unions, maximum **2k** elements belong to such unions.

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

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

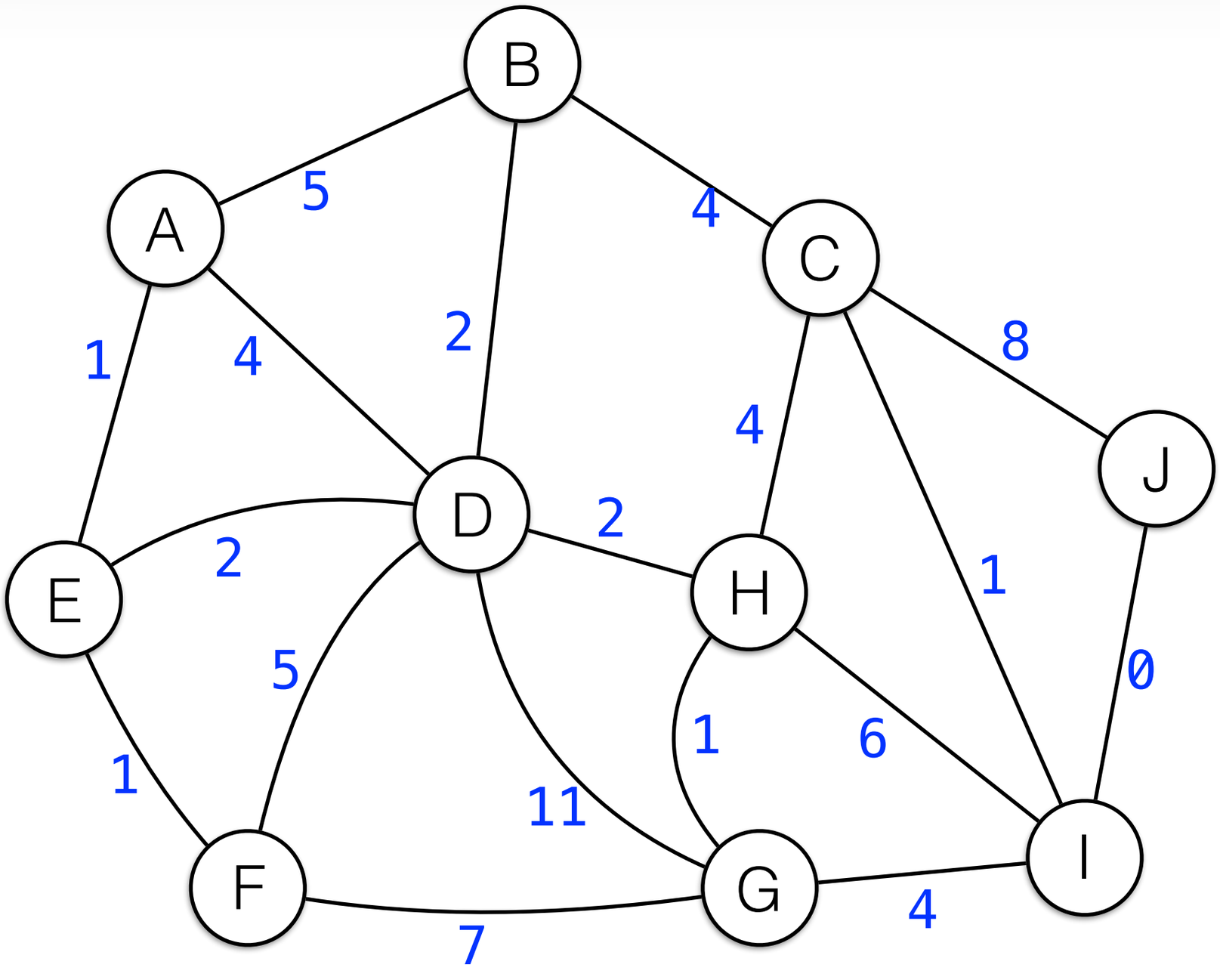
Changing **2k** elements, each is updated maximum **`log(2k)`** times, so the complexity is **`2klog(2k)`**.

MST



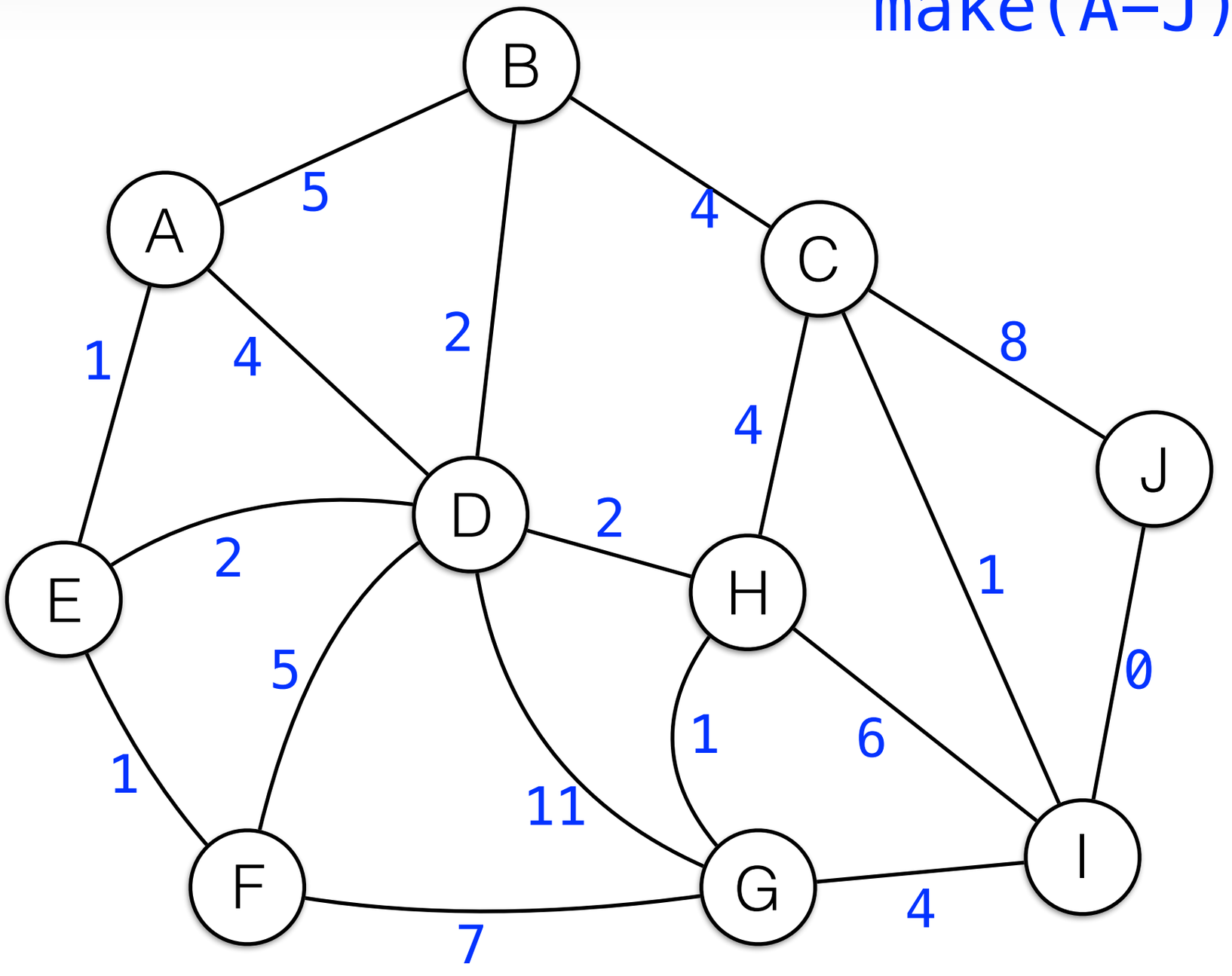
Kruskal MST

	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



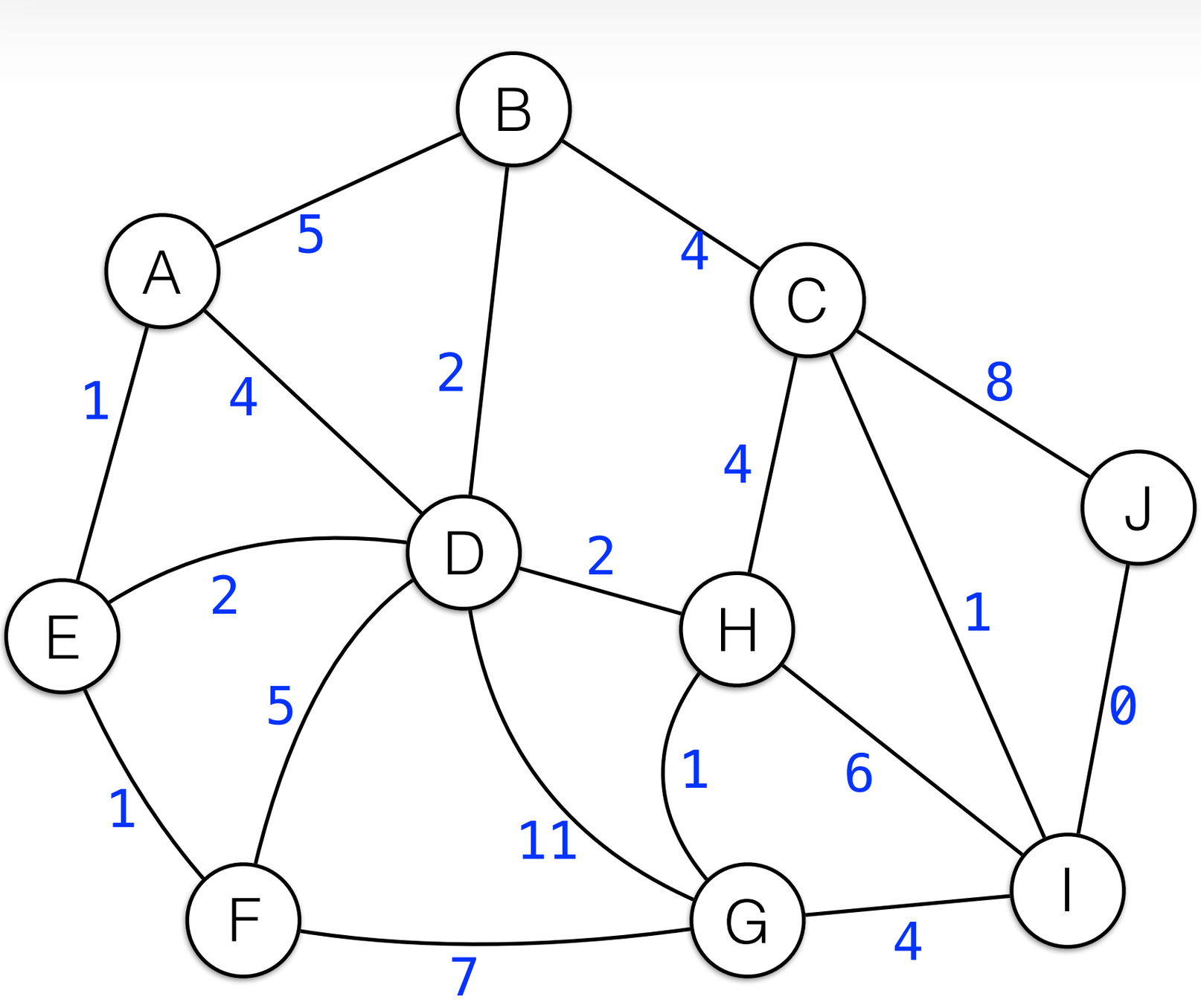
Kruskal MST

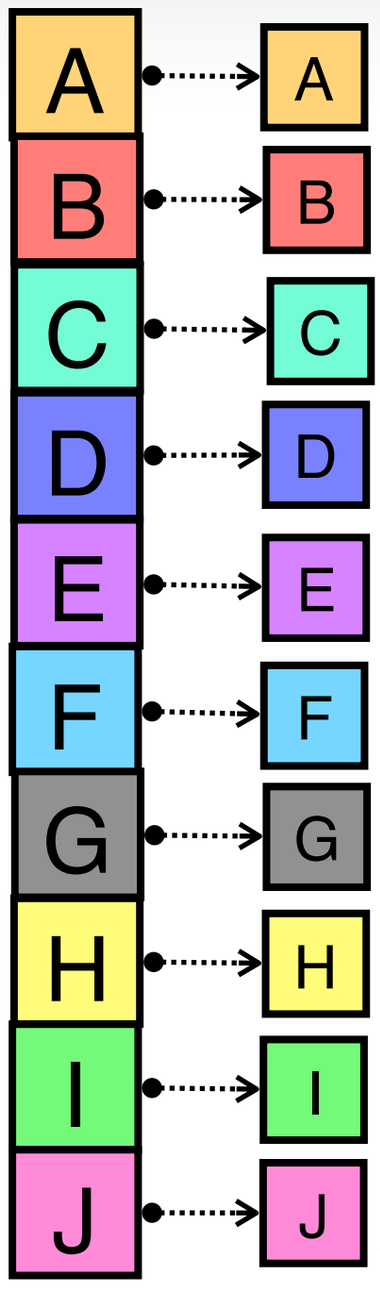
	I-J	A-E	C-I	E-F	G-H	B-D	C-J	D-E	D-H	A-D	B-C	C-H	G-I	A-B	D-F	H-I	F-G	D-G
d	0	1	1	1	1	2	2	2	2	4	4	4	4	5	5	6	7	11



Kruskal MST

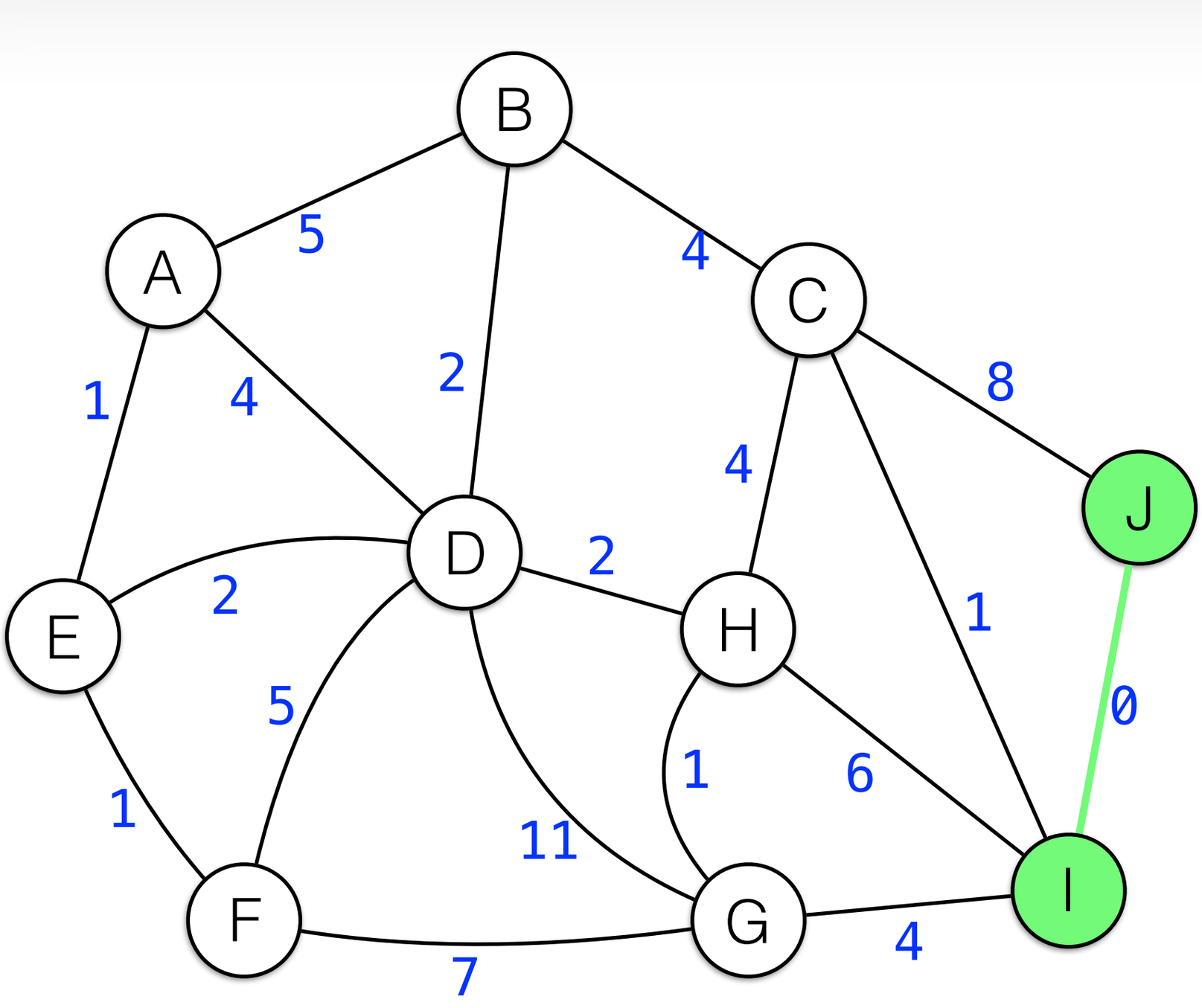
	I-J	A-E	C-I	E-F	G-H	B-D	C-J	D-E	D-H	A-D	B-C	C-H	G-I	A-B	D-F	H-I	F-G	D-G
d	0	1	1	1	1	2	2	2	2	4	4	4	4	5	5	6	7	11

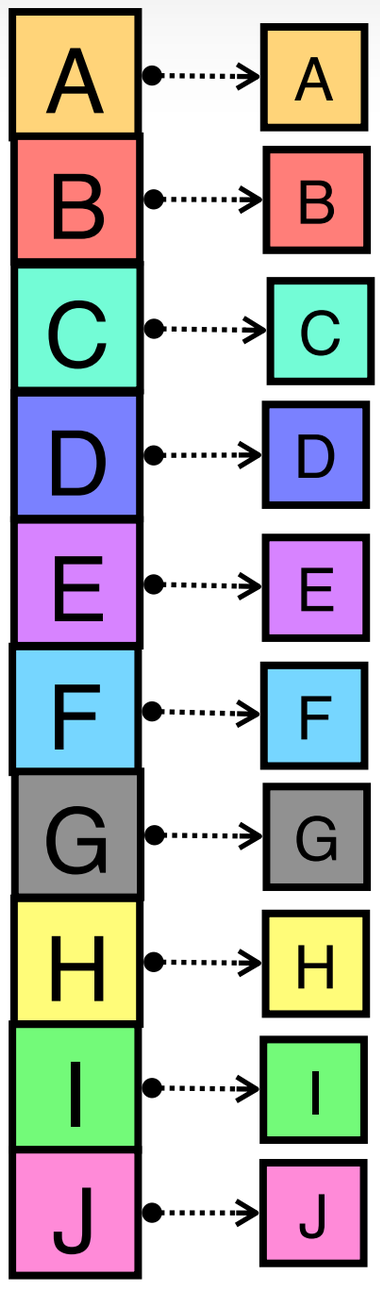




Kruskal MST

	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

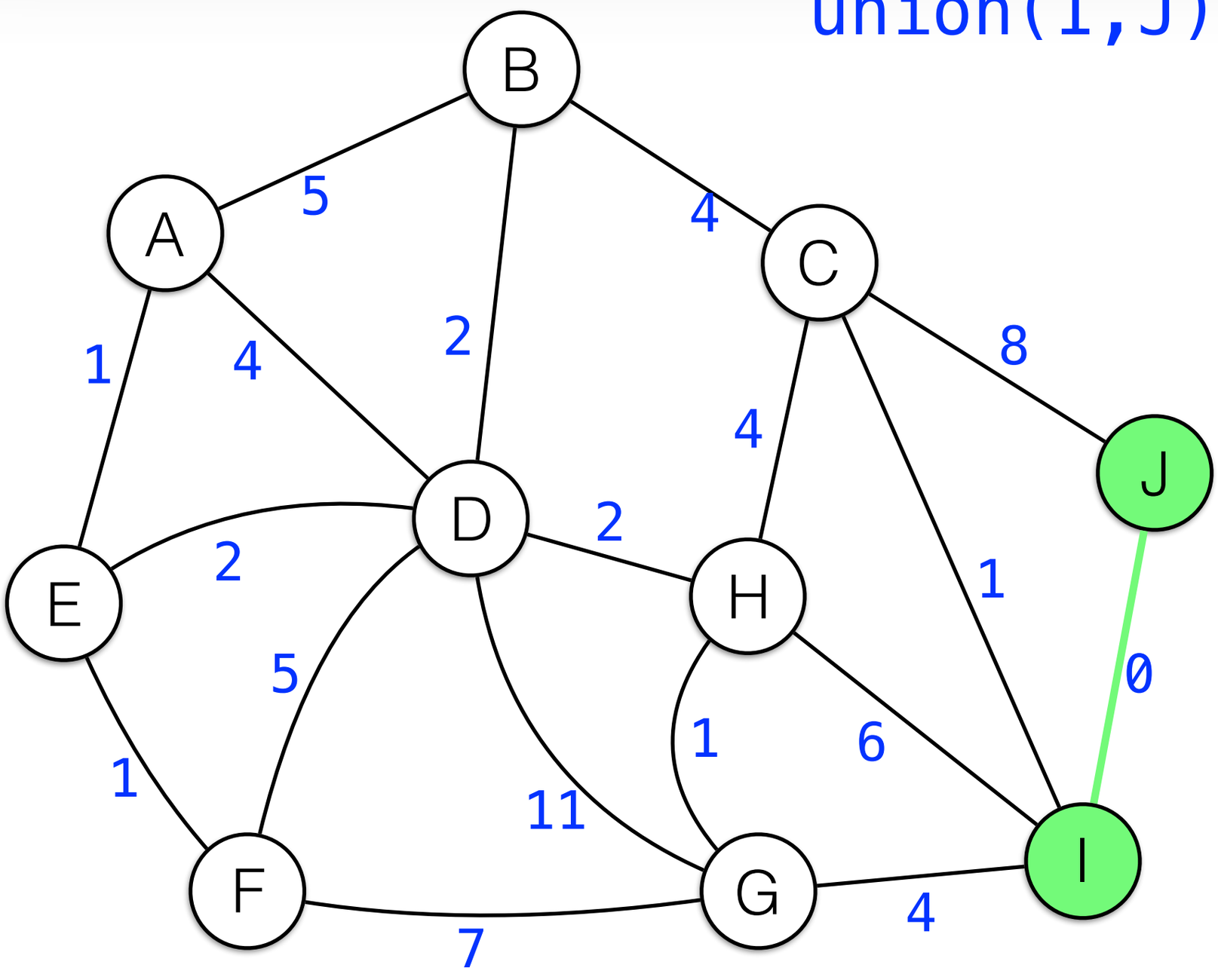


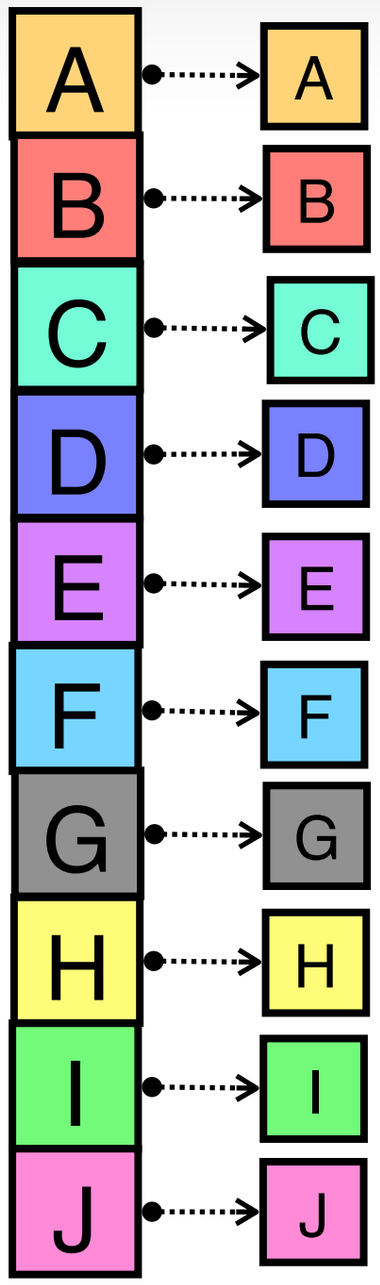


Kruskal MST

	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

union(I,J)

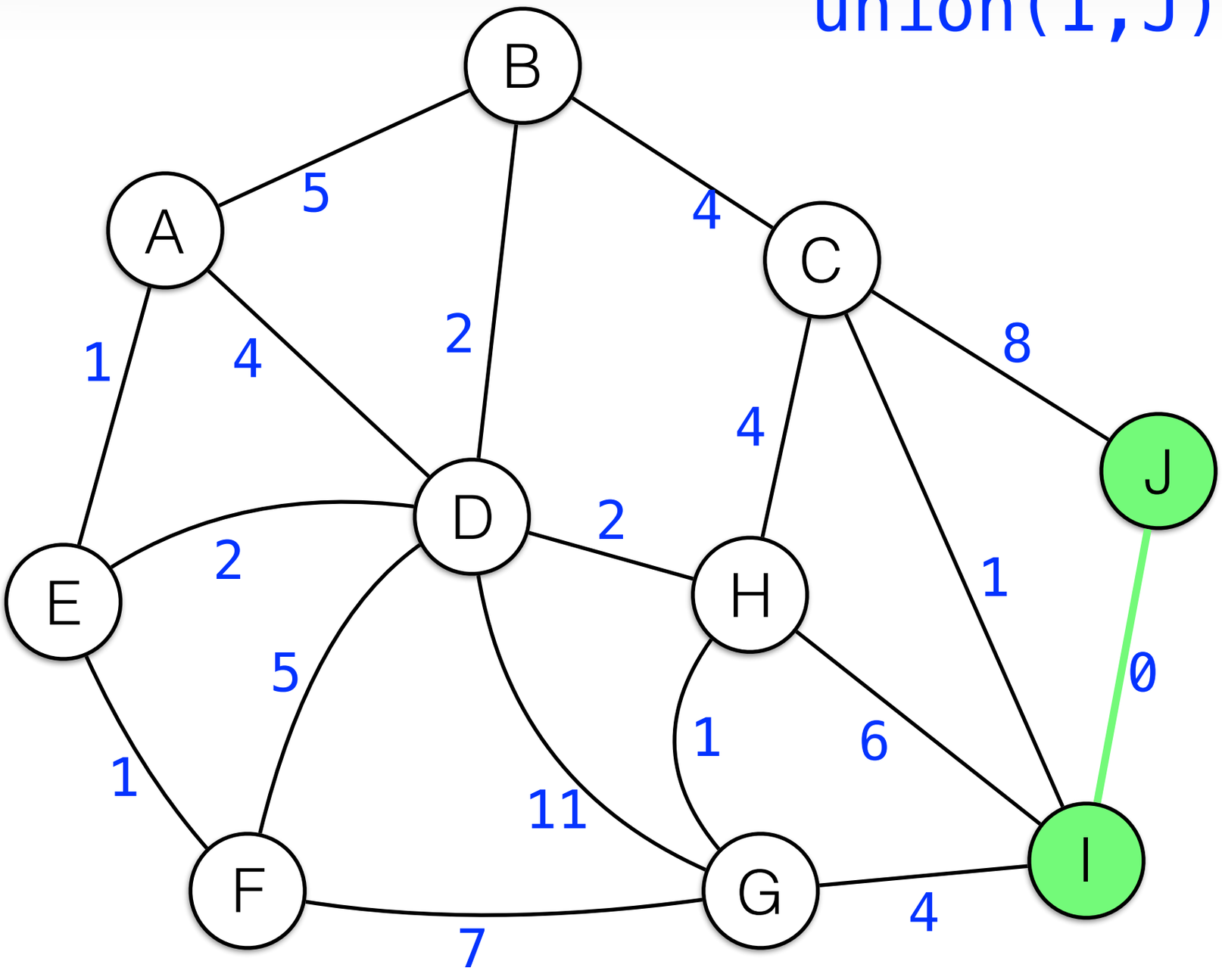


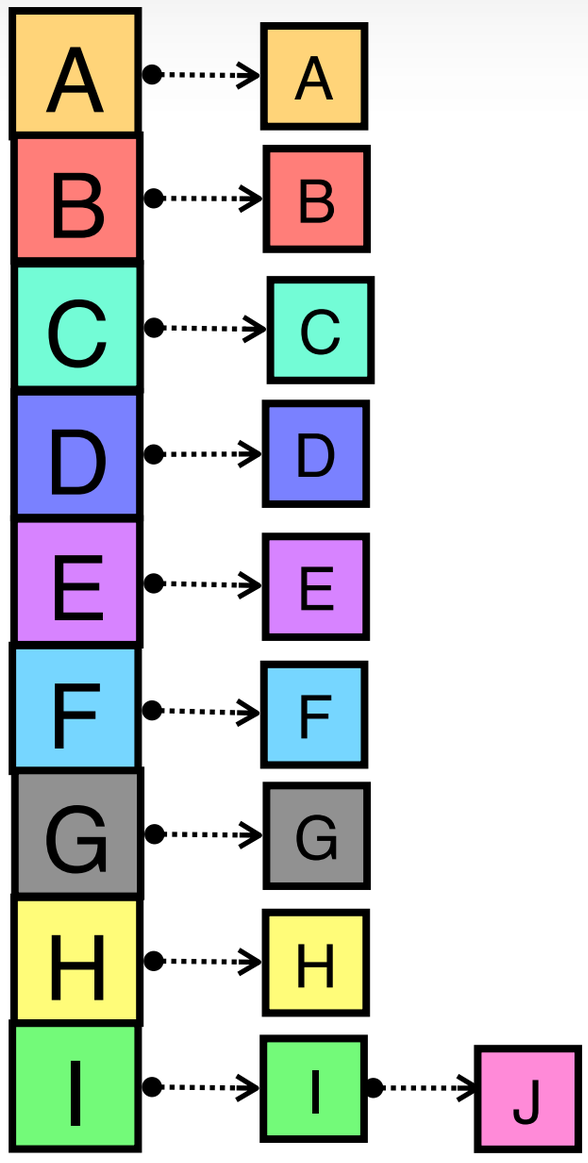


Kruskal MST

	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

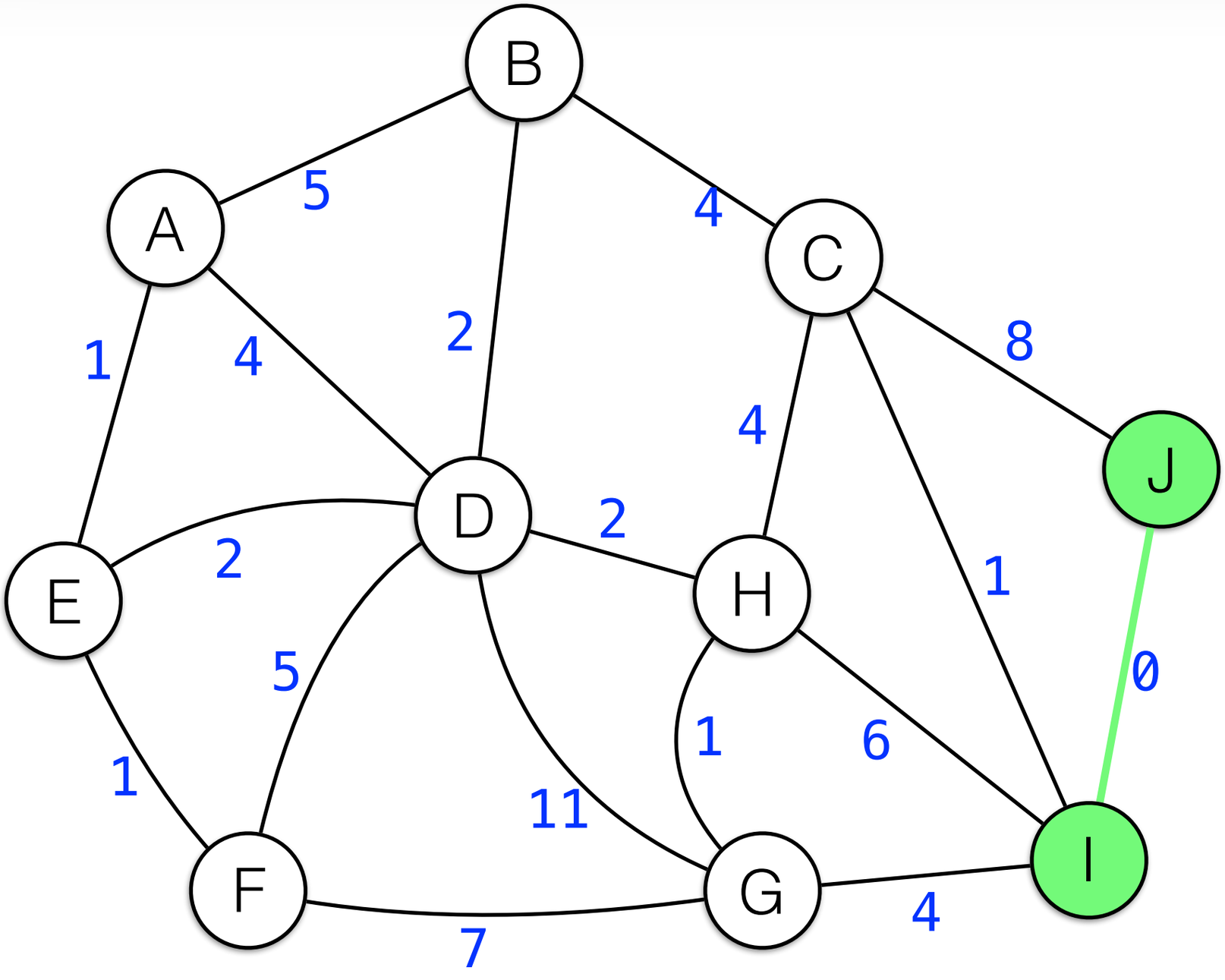
union(I,J)

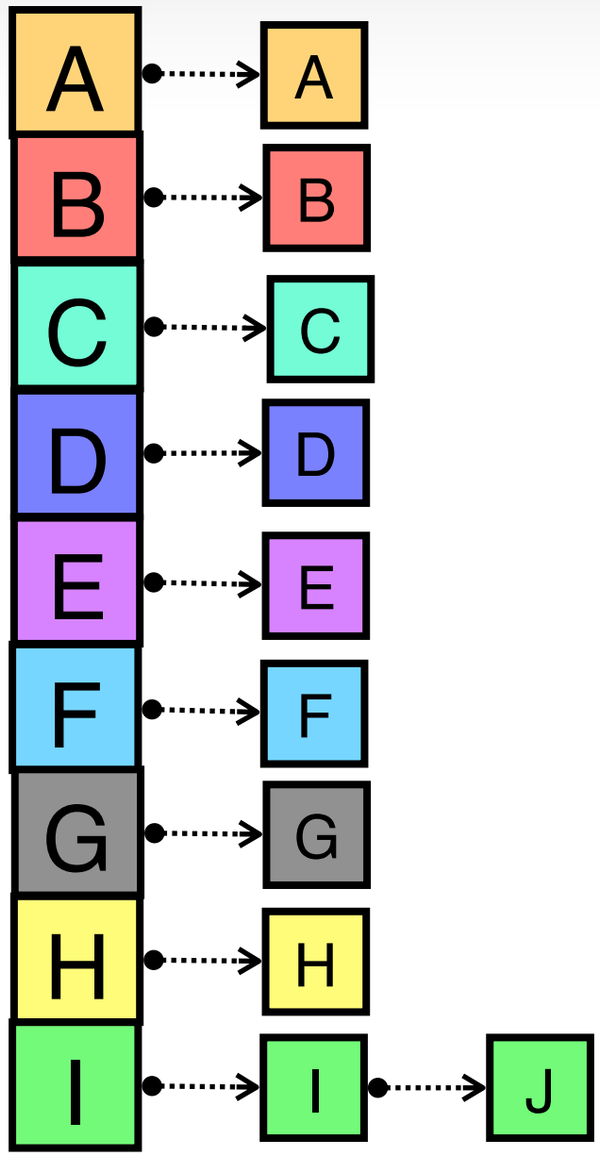




Kruskal MST

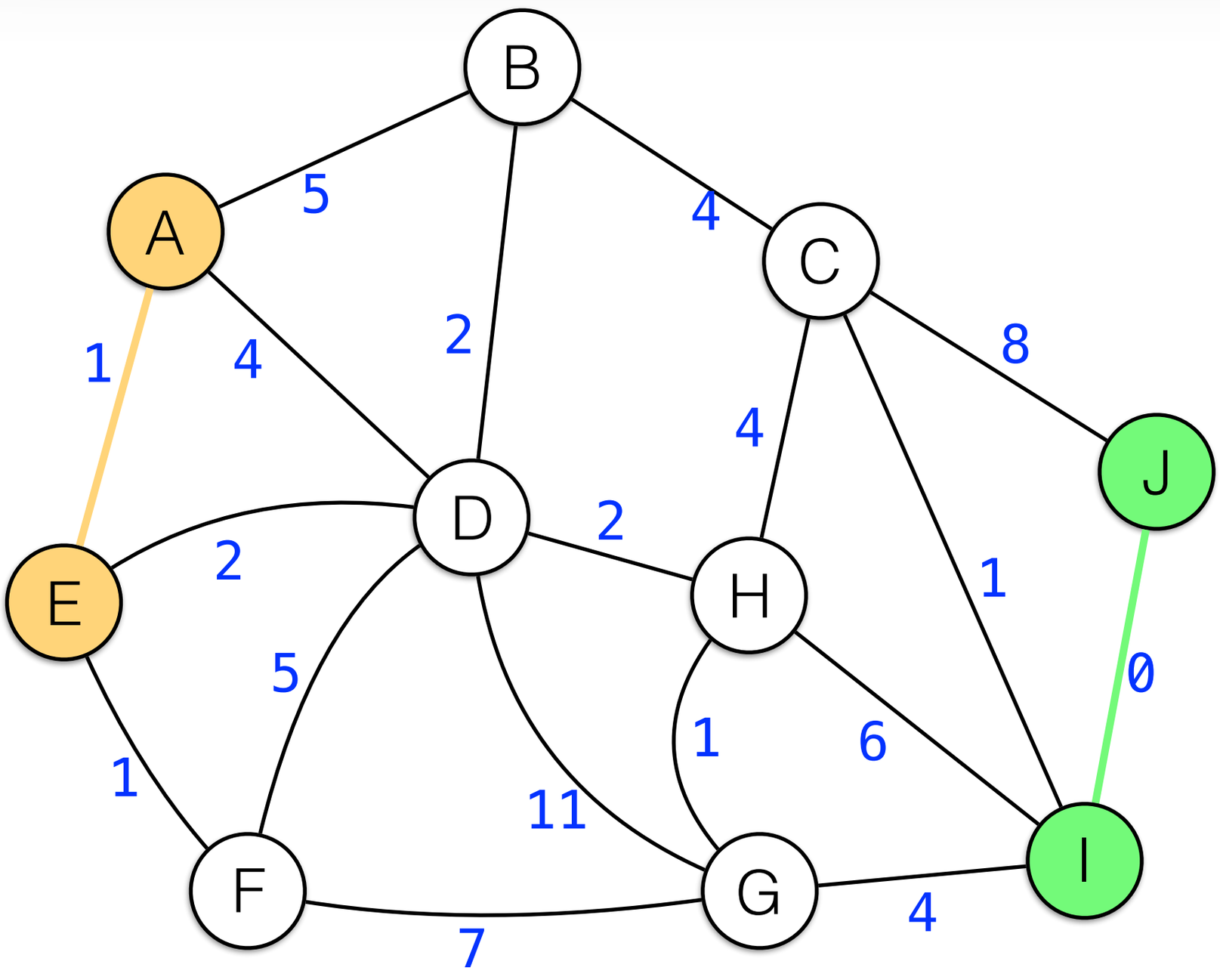
	I-J	A-E	C-I	E-F	G-H	B-D	C-J	D-E	D-H	A-D	B-C	C-H	G-I	A-B	D-F	H-I	F-G	D-G
d	0	1	1	1	1	2	2	2	2	4	4	4	4	5	5	6	7	11

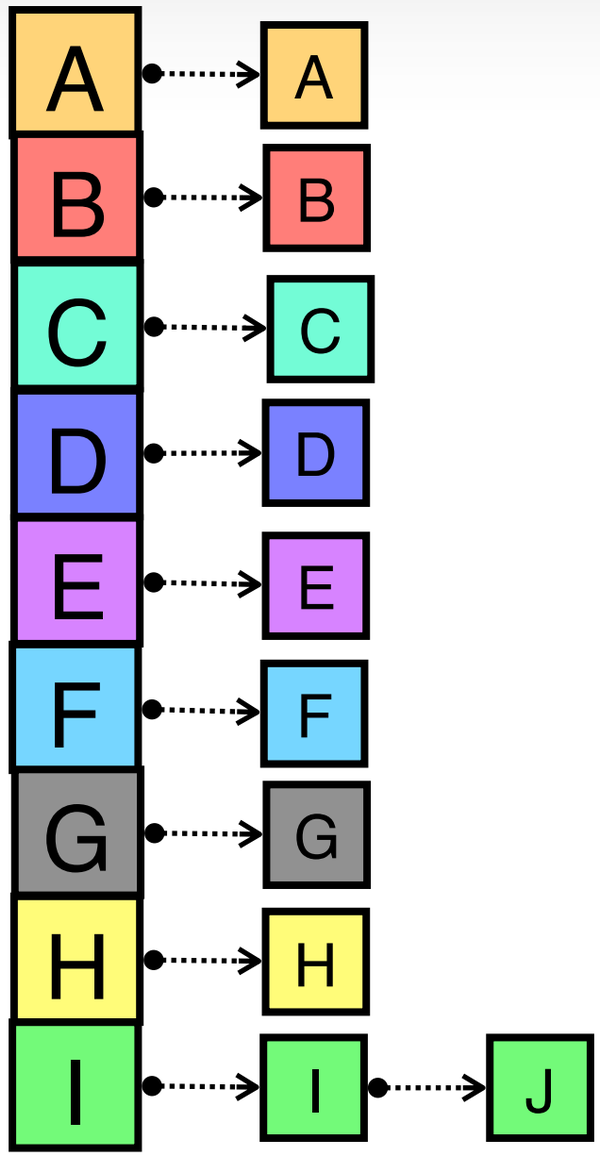




Kruskal MST

	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

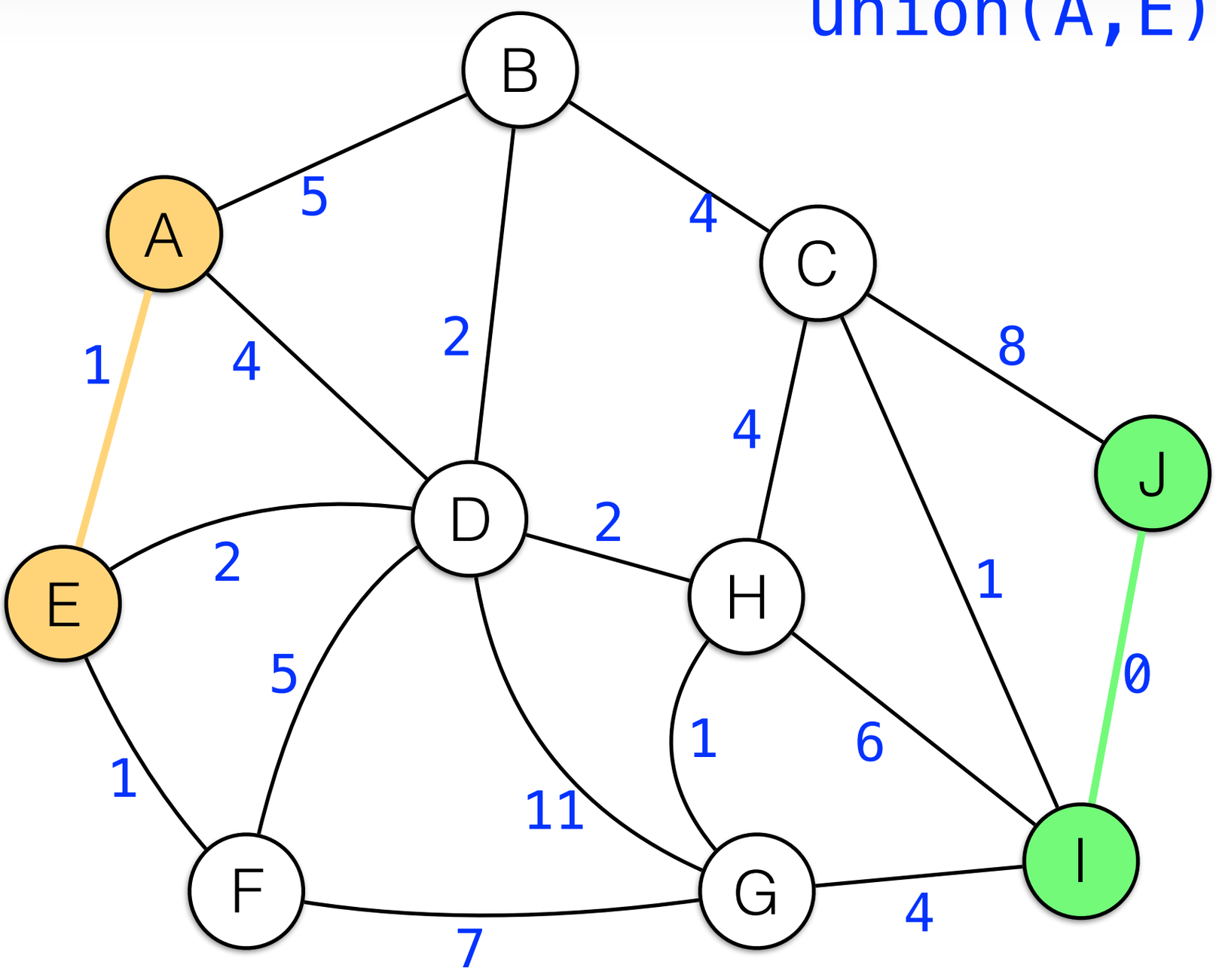


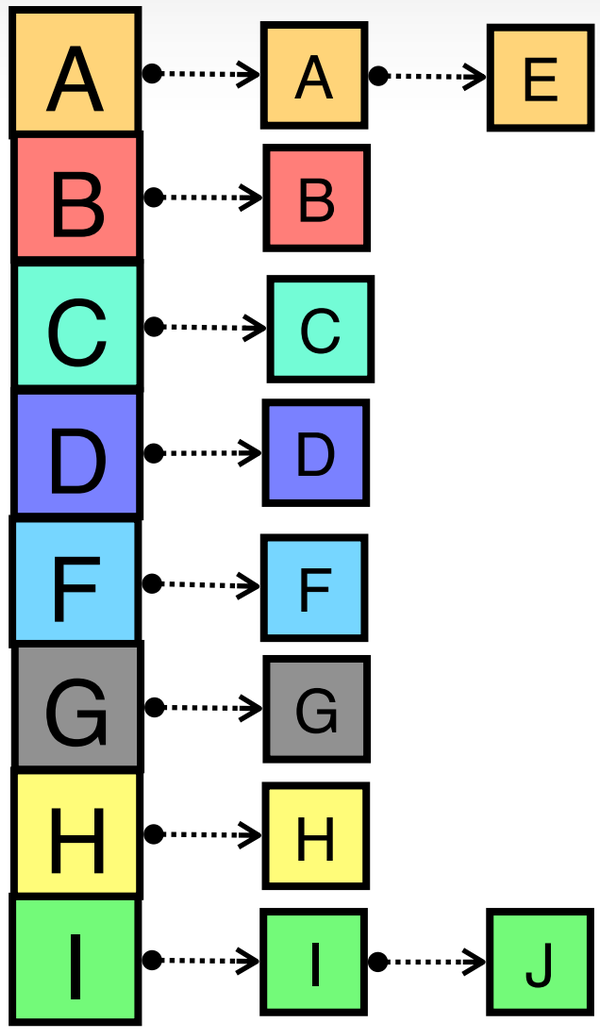


Kruskal MST

	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

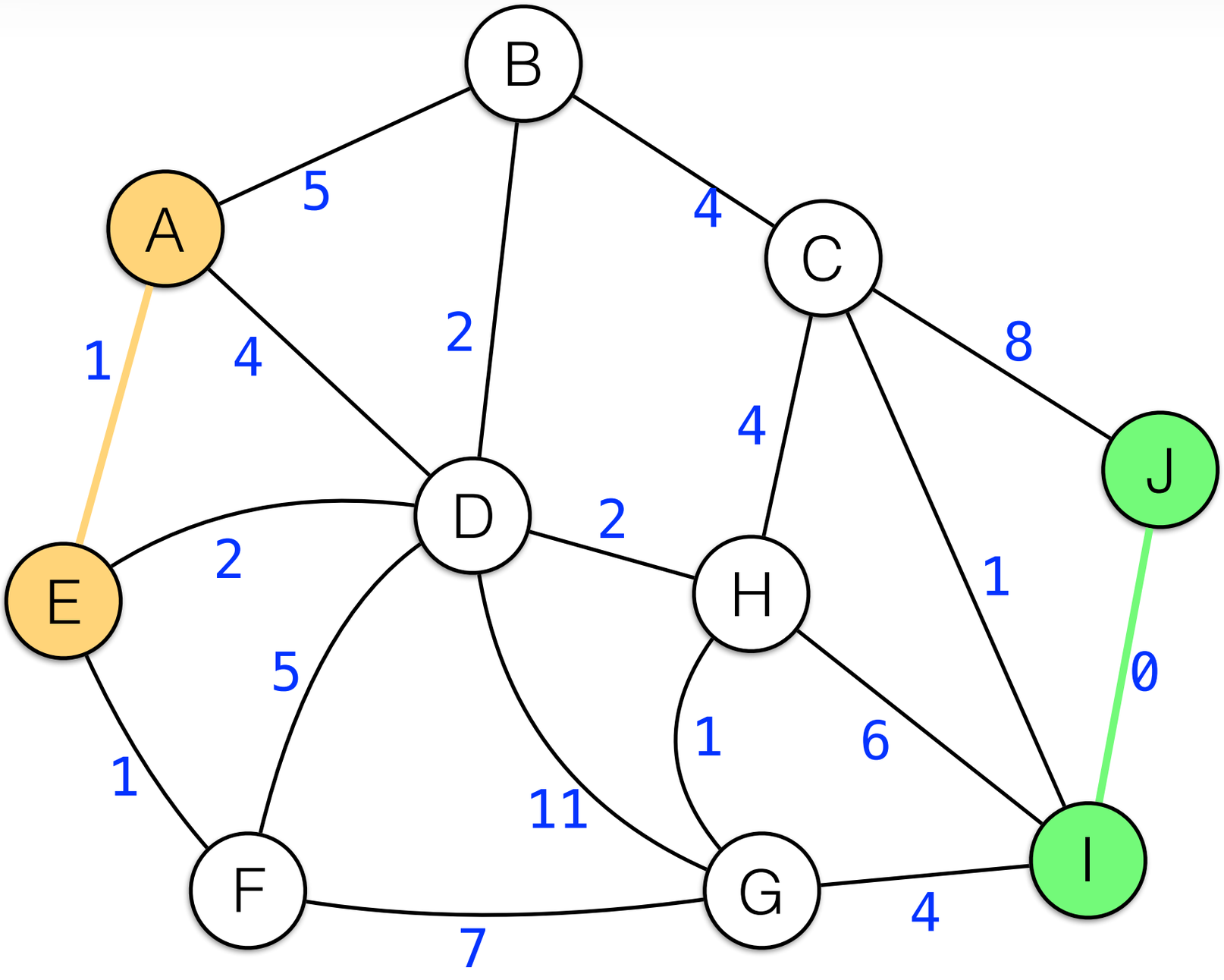
union(A,E)

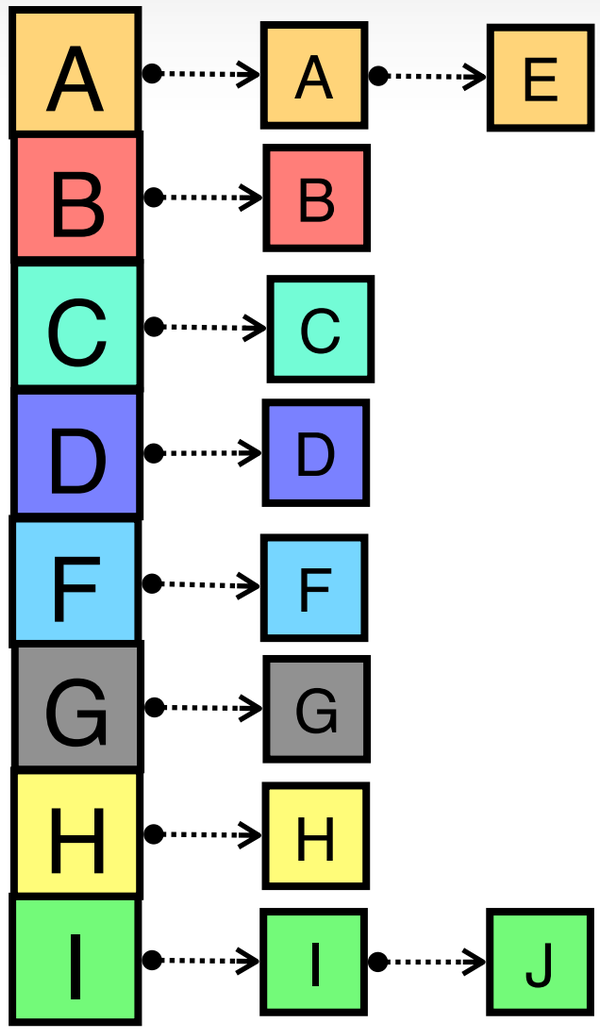




Kruskal MST

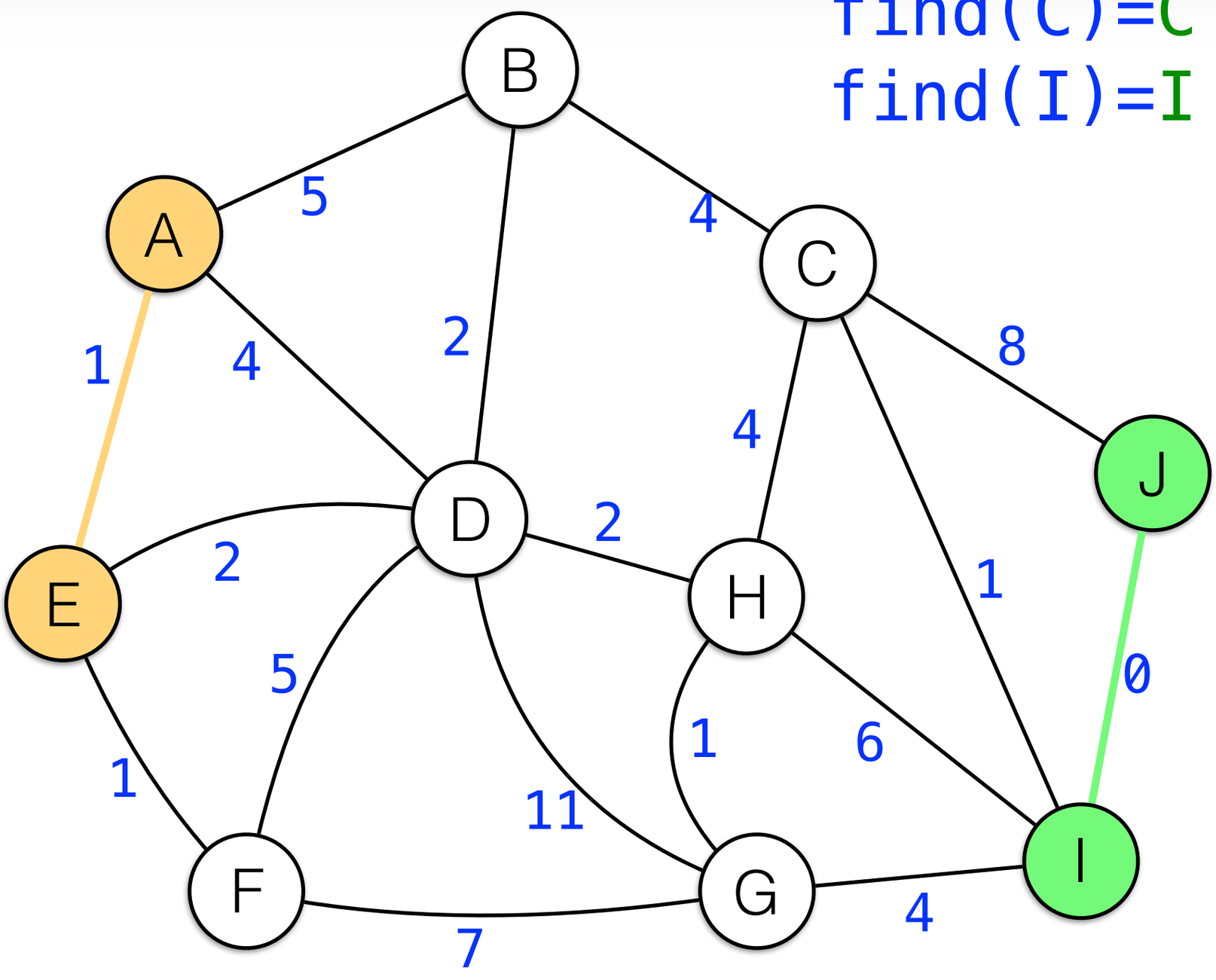
	I-J	A-E	C-I	E-F	G-H	B-D	C-J	D-E	D-H	A-D	B-C	C-H	G-I	A-B	D-F	H-I	F-G	D-G
d	0	1	1	1	1	2	2	2	2	4	4	4	4	5	5	6	7	11



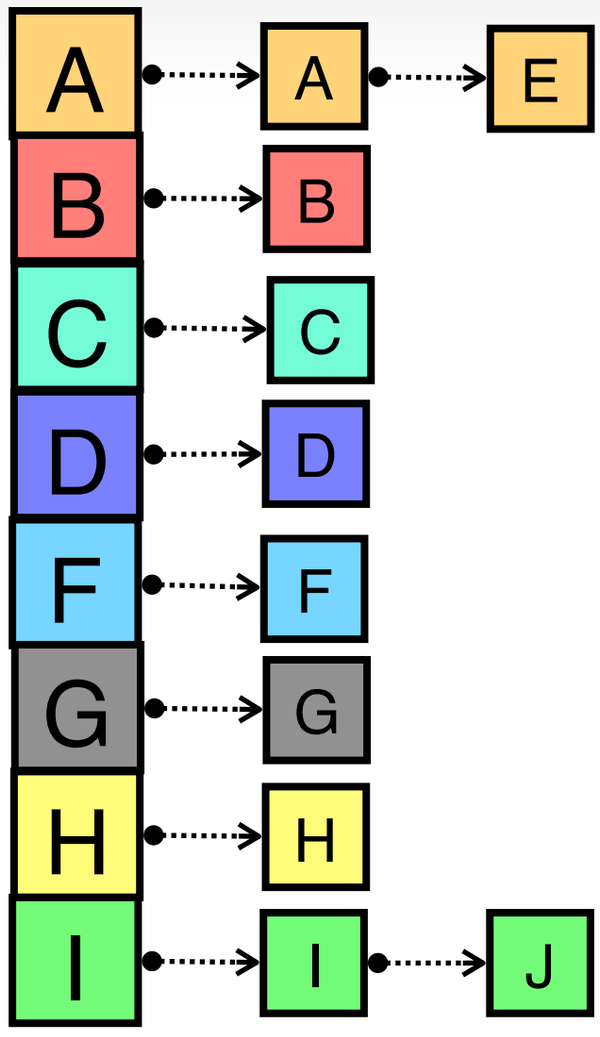


Kruskal MST

	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

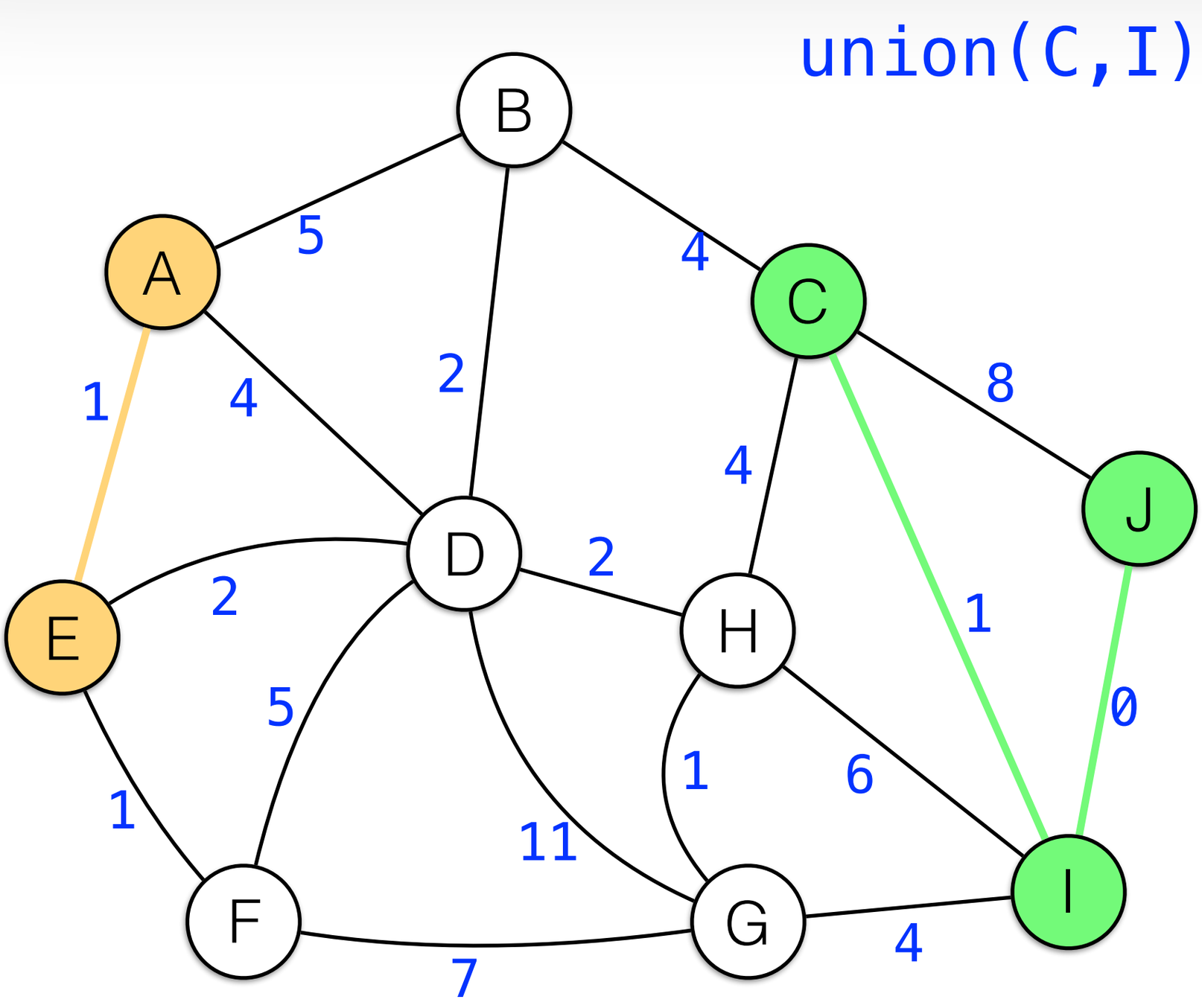



```
find(C)=C
find(I)=I
```

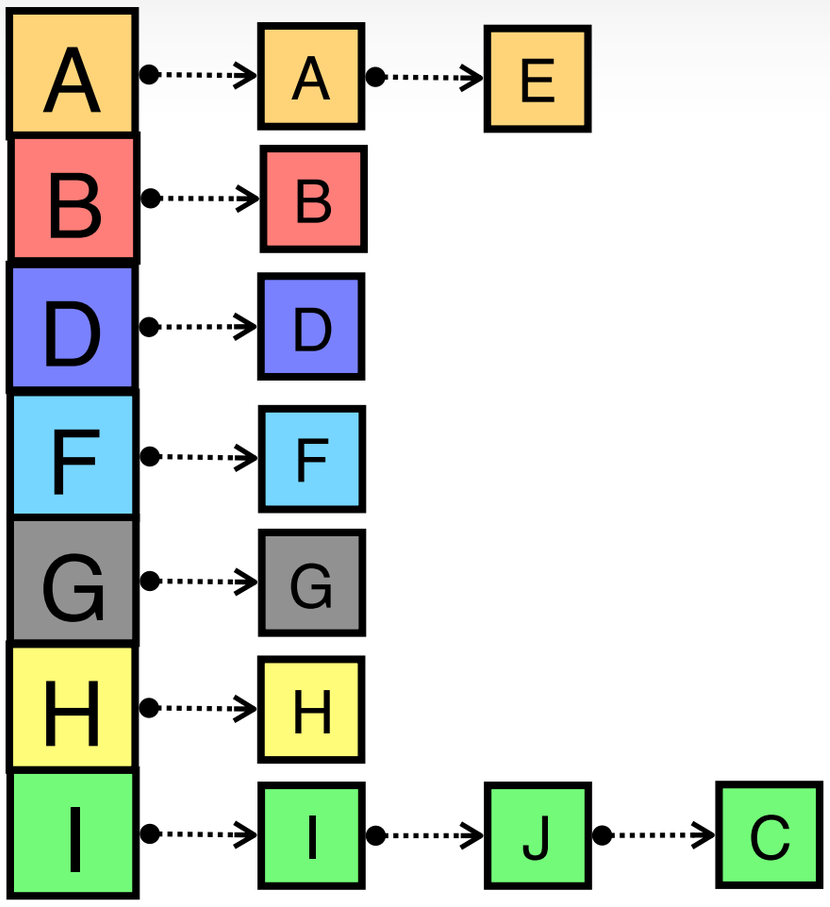


Kruskal MST

	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

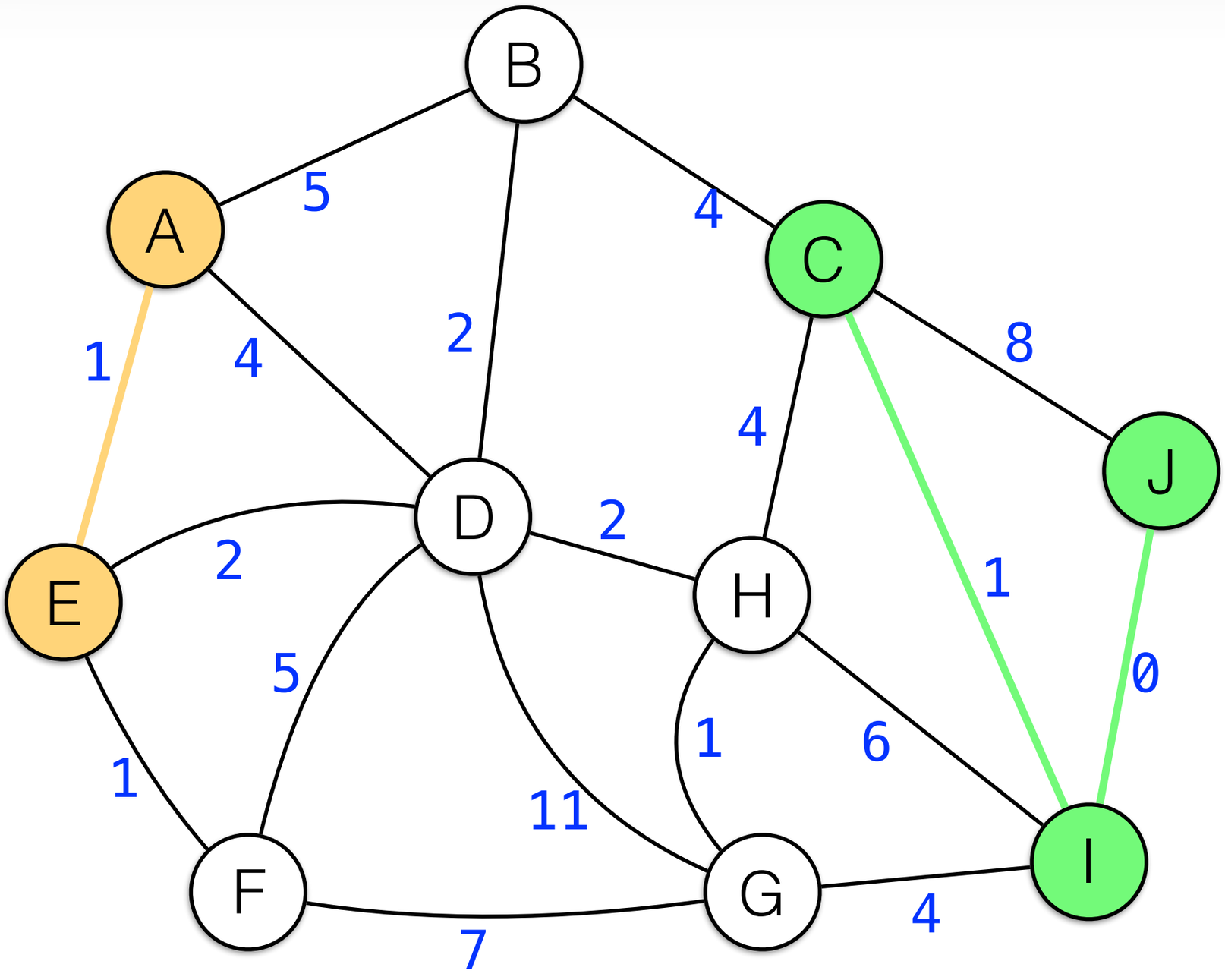


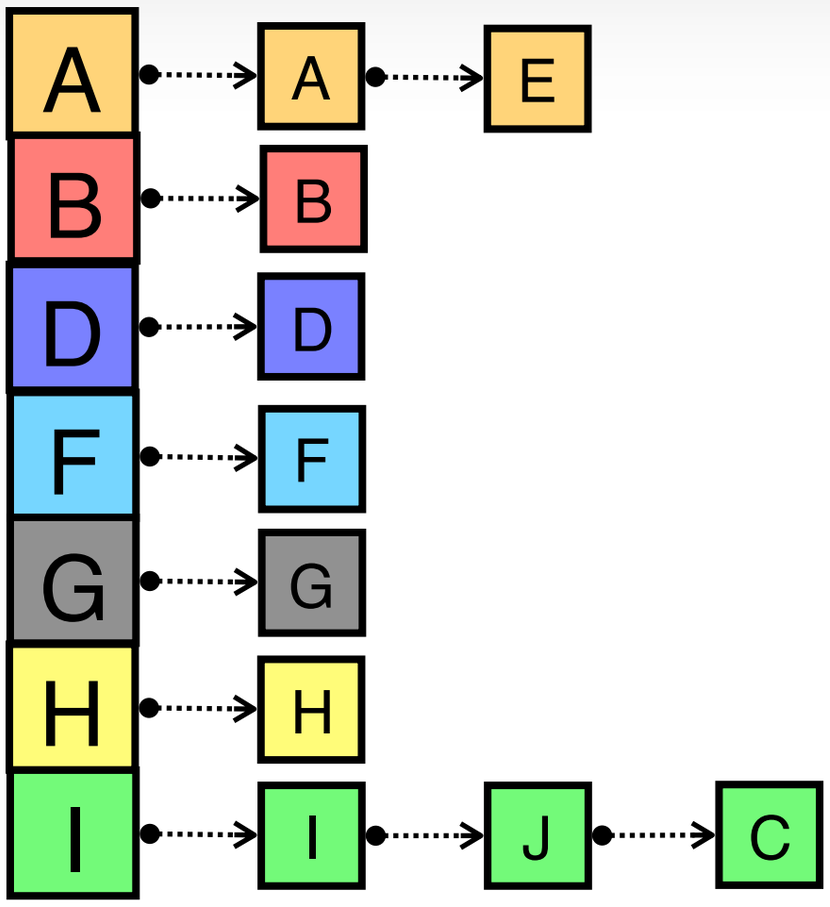
union(C,I)



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

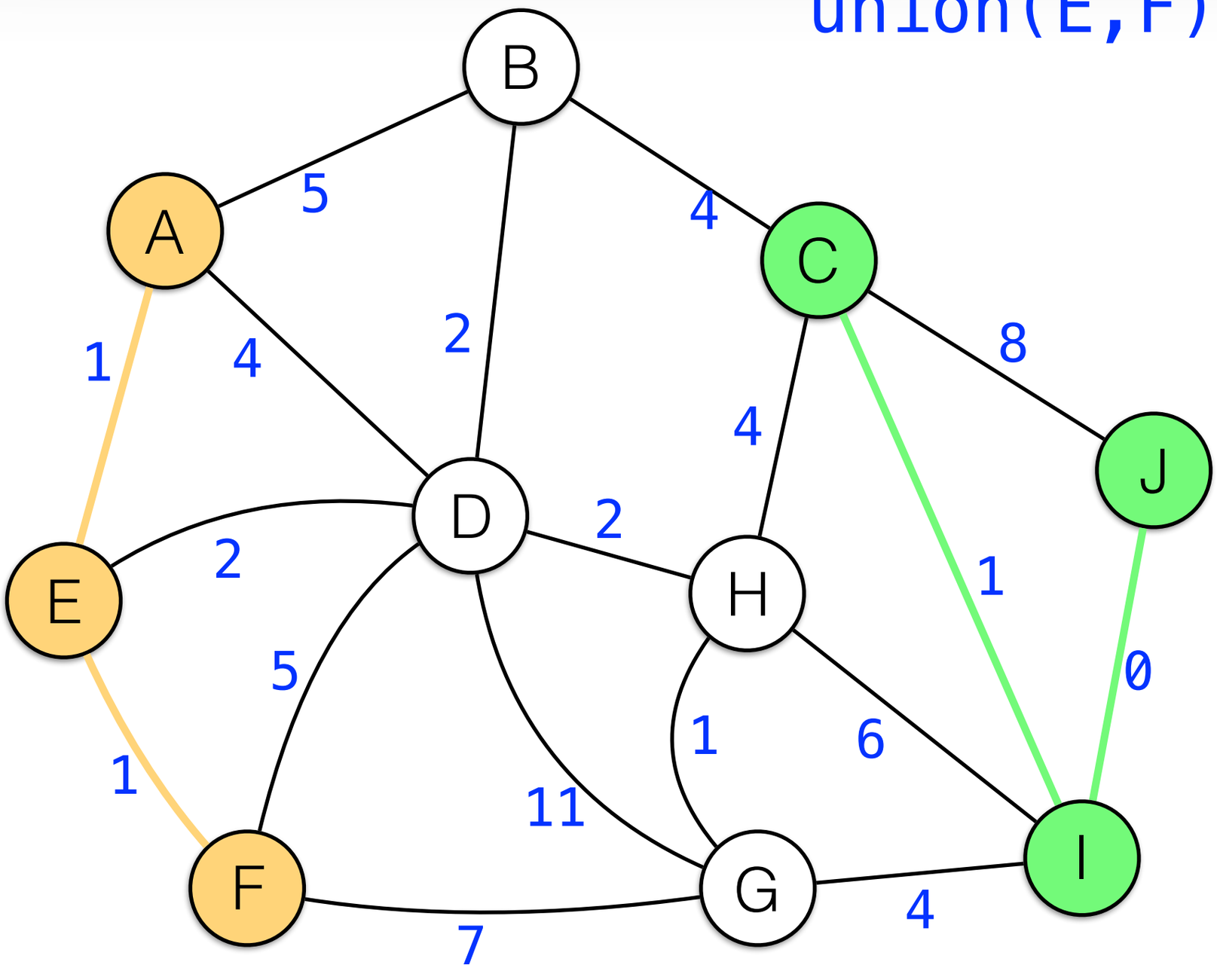


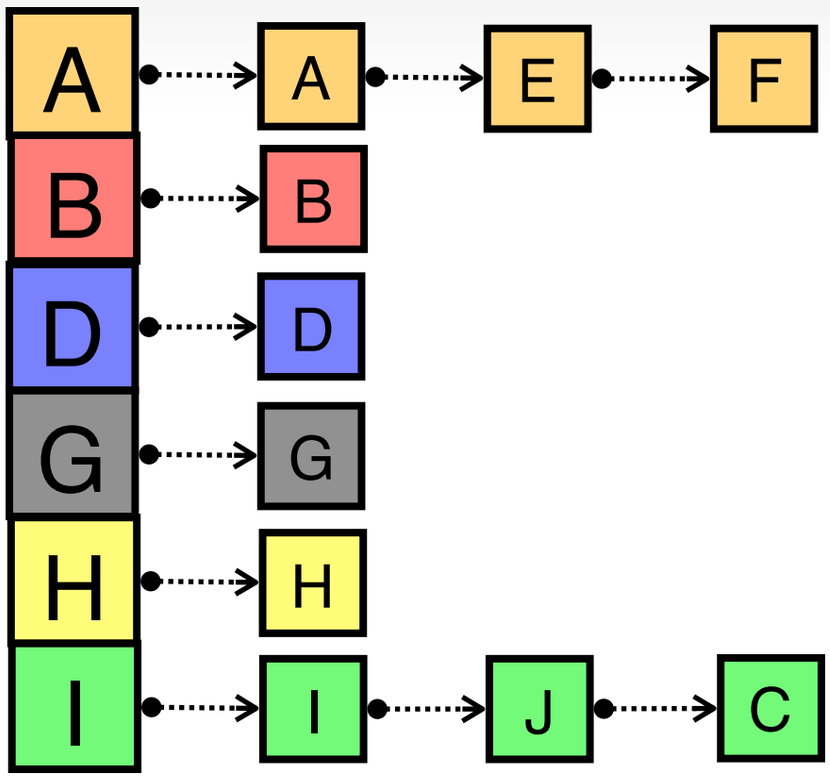


Kruskal MST

	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

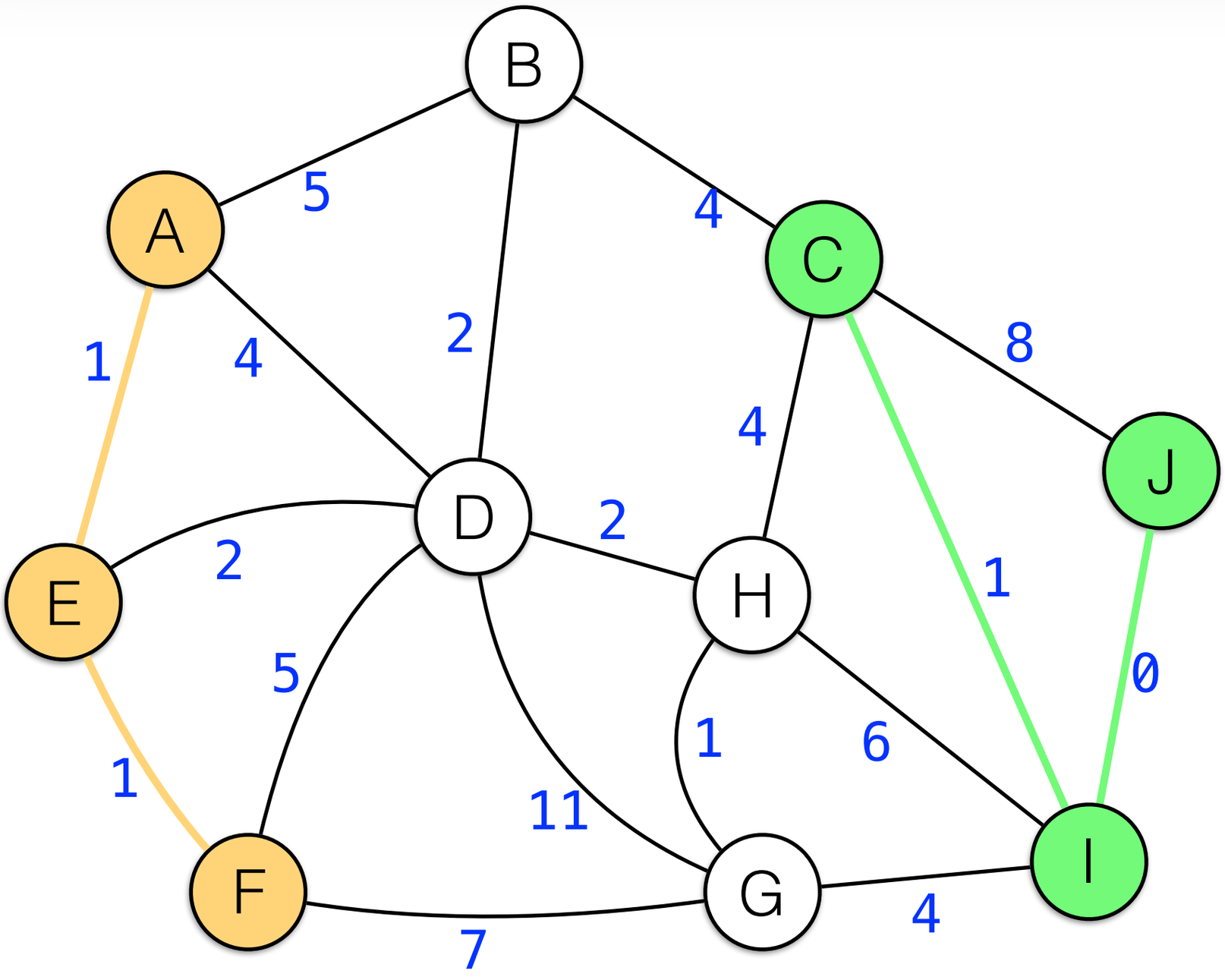
union(E,F)

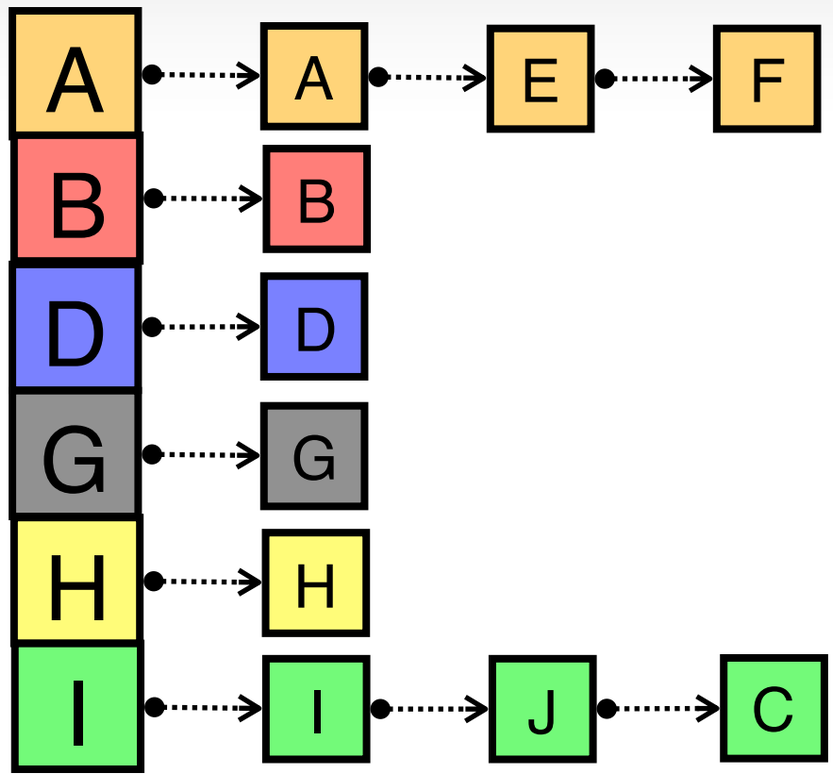




Kruskal MST

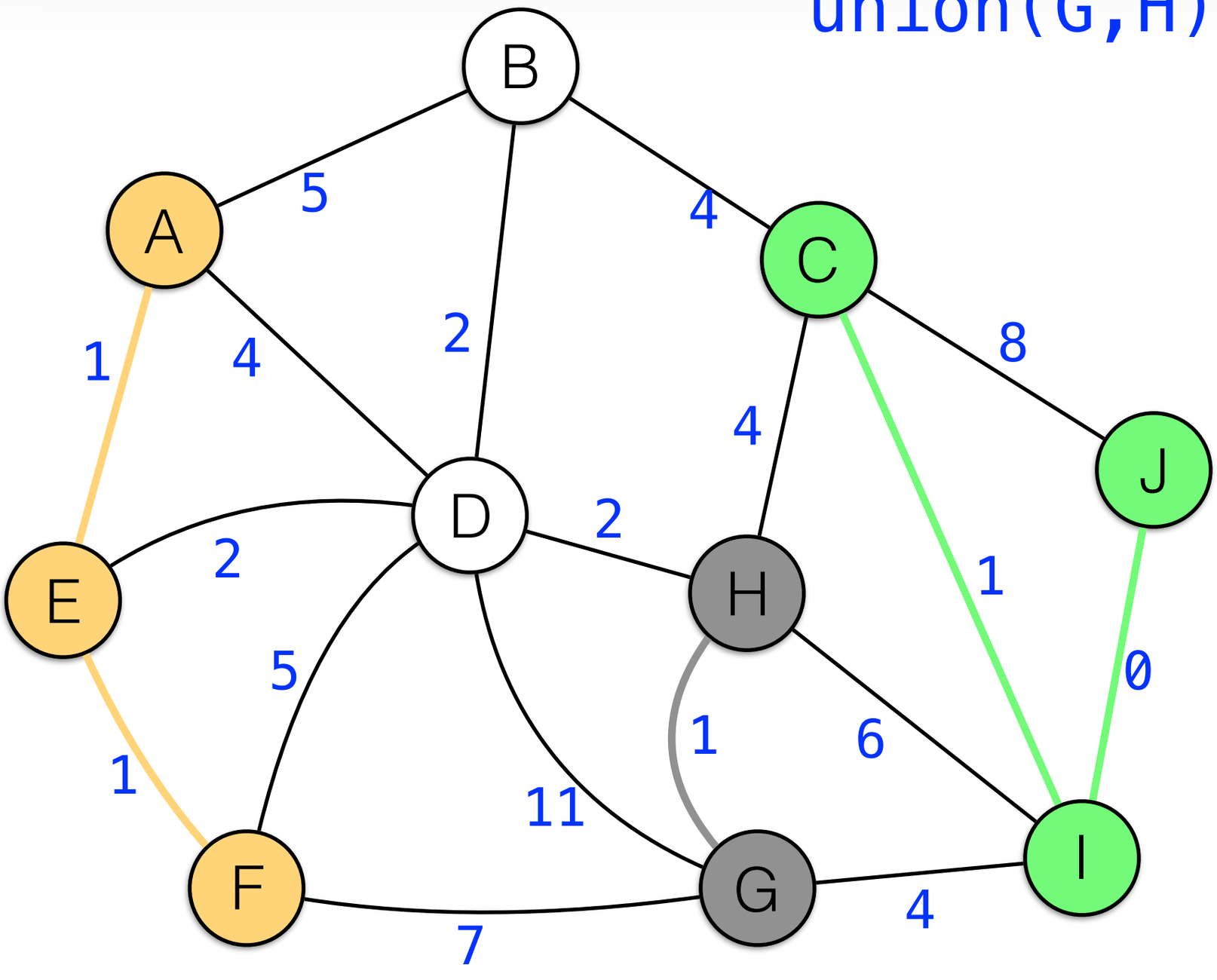
	I-J	A-E	C-I	E-F	G-H	B-D	C-J	D-E	D-H	A-D	B-C	C-H	G-I	A-B	D-F	H-I	F-G	D-G
d	0	1	1	1	1	2	2	2	2	4	4	4	4	5	5	6	7	11



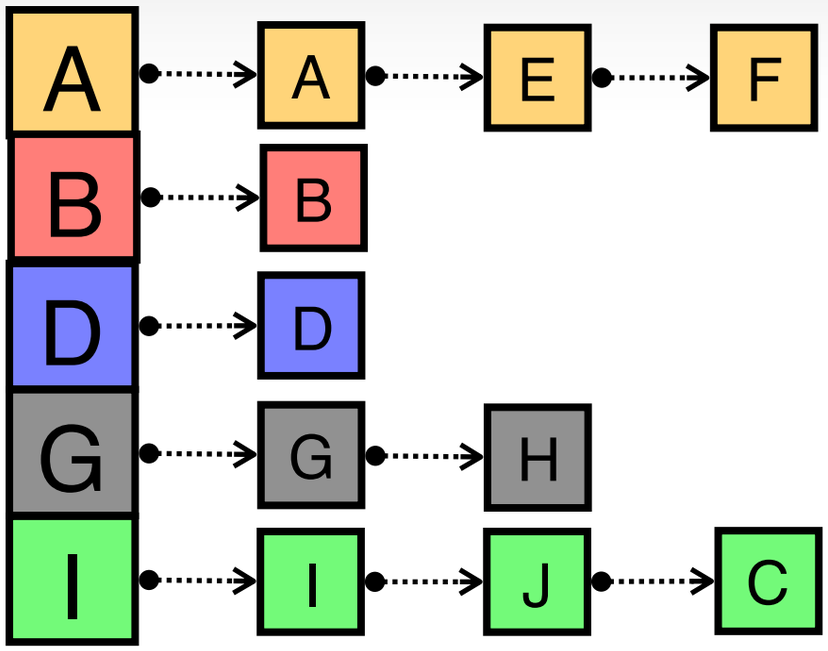


Kruskal MST

	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

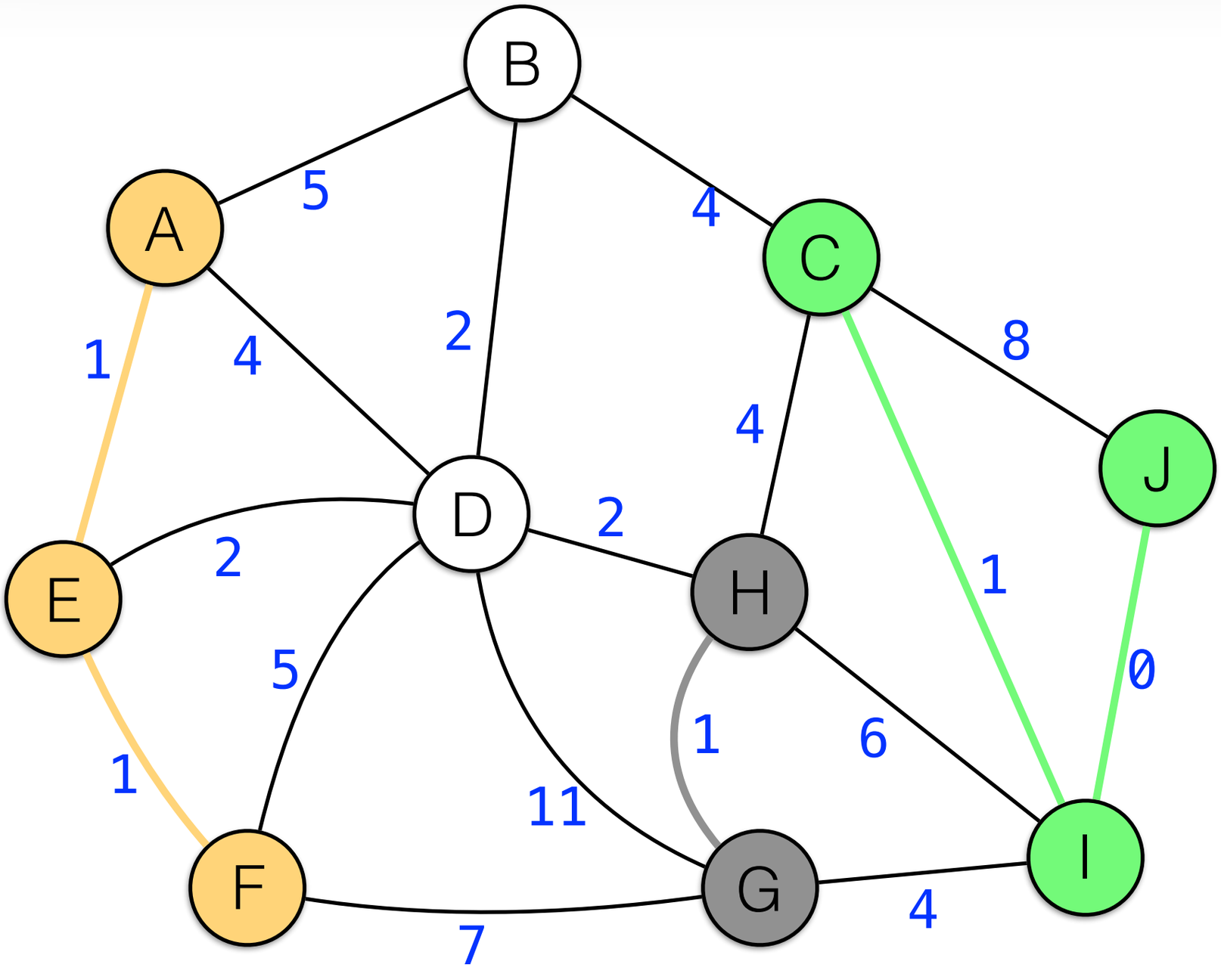


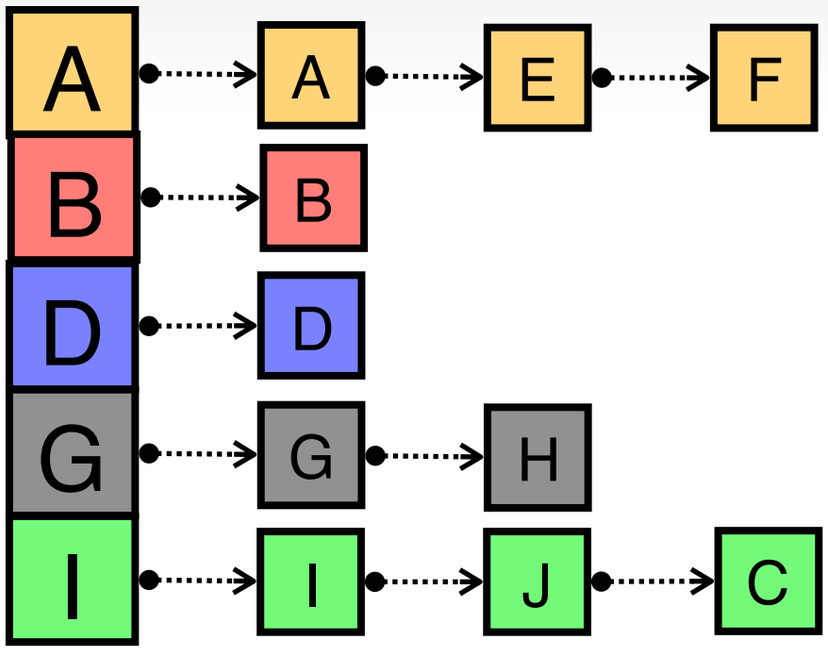
union(G,H)



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

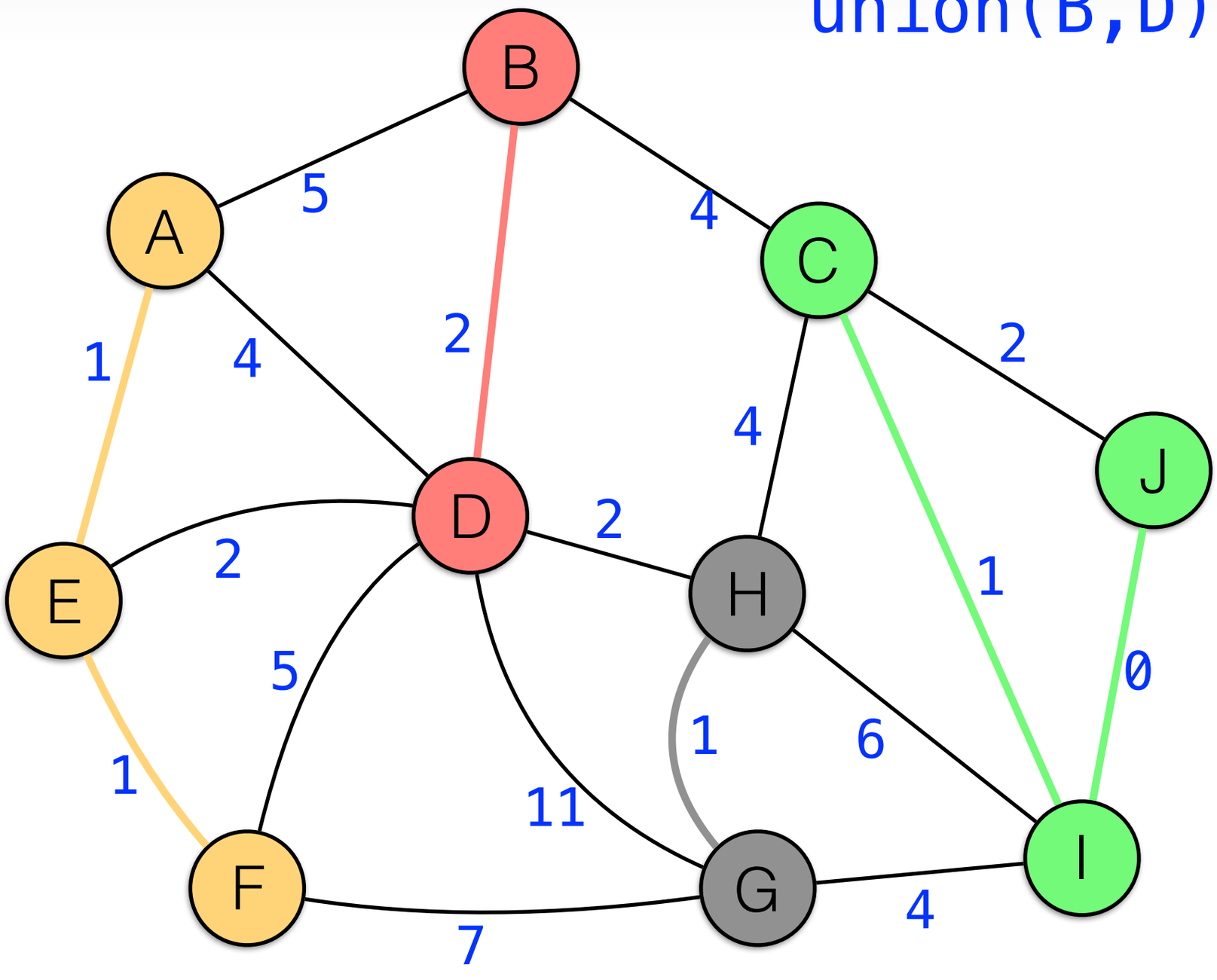


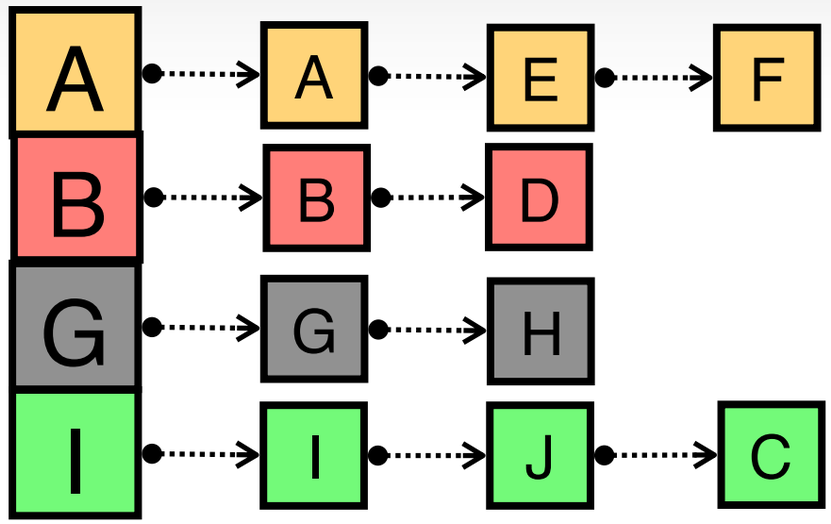


Kruskal MST

	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

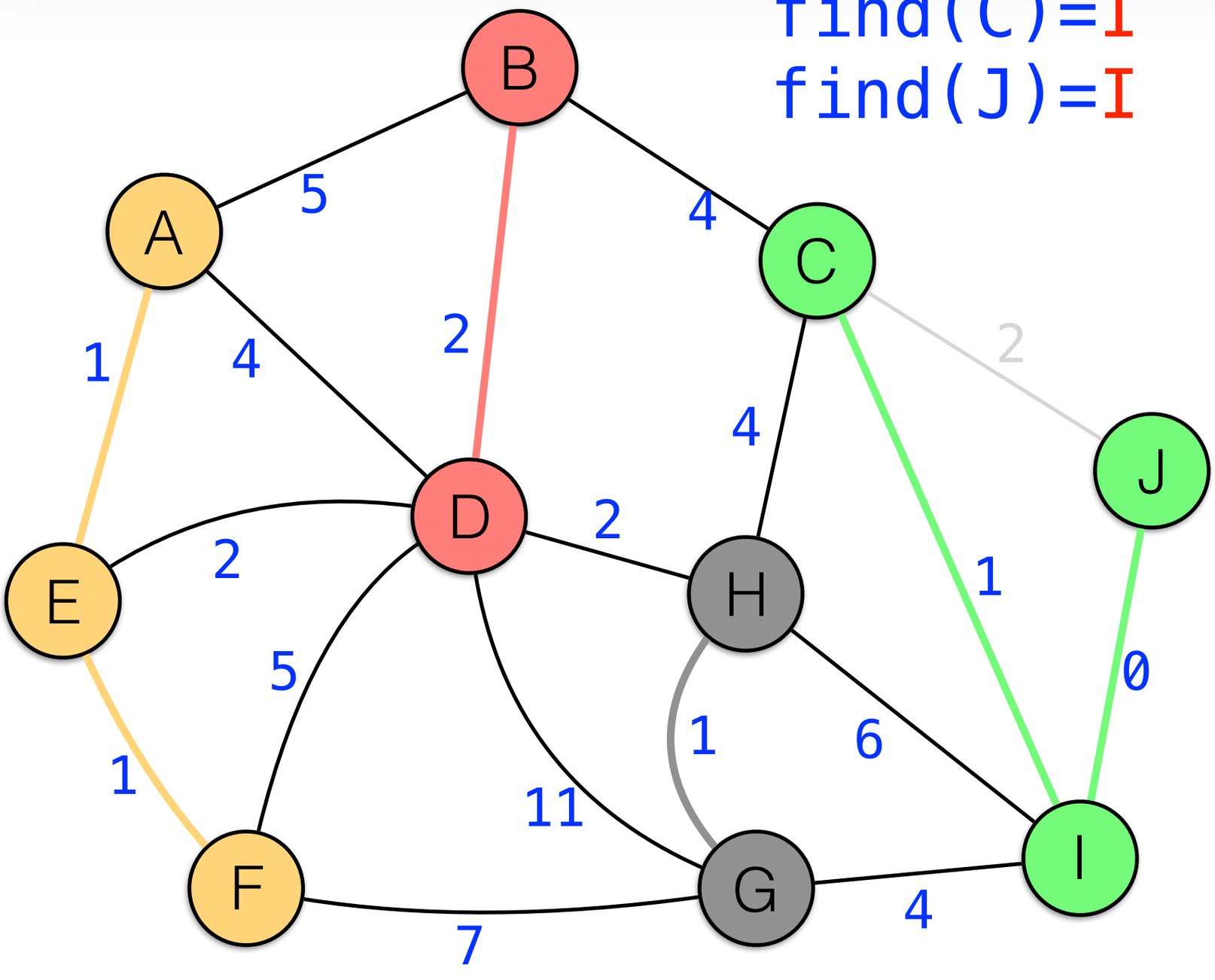
union(B,D)



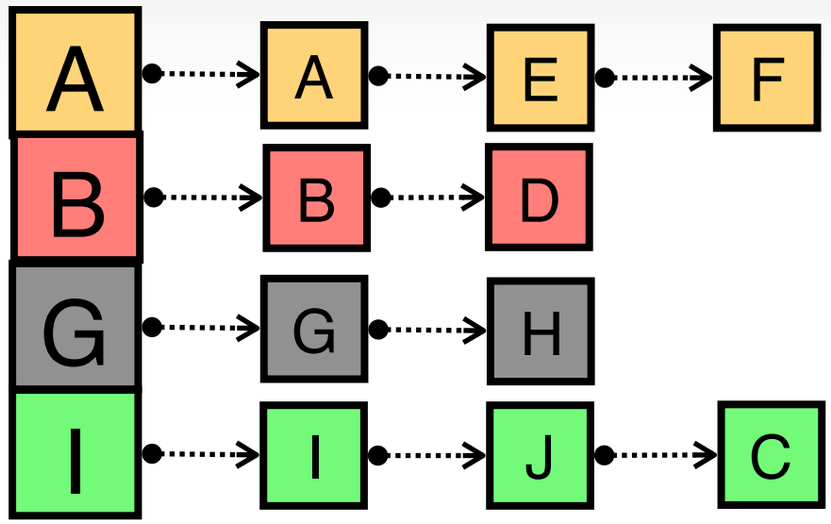


Kruskal MST

	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

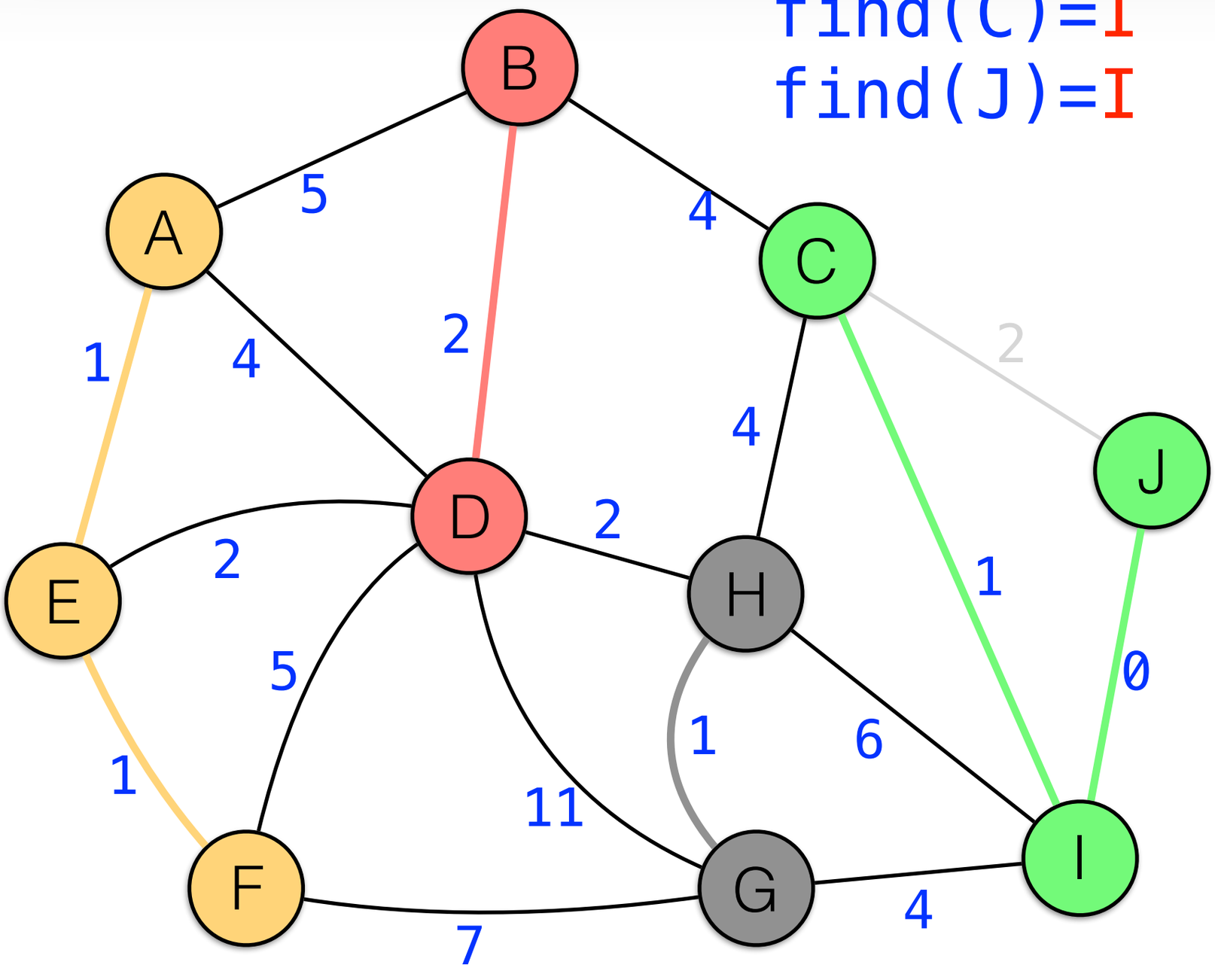



```
find(C)=I
find(J)=I
```

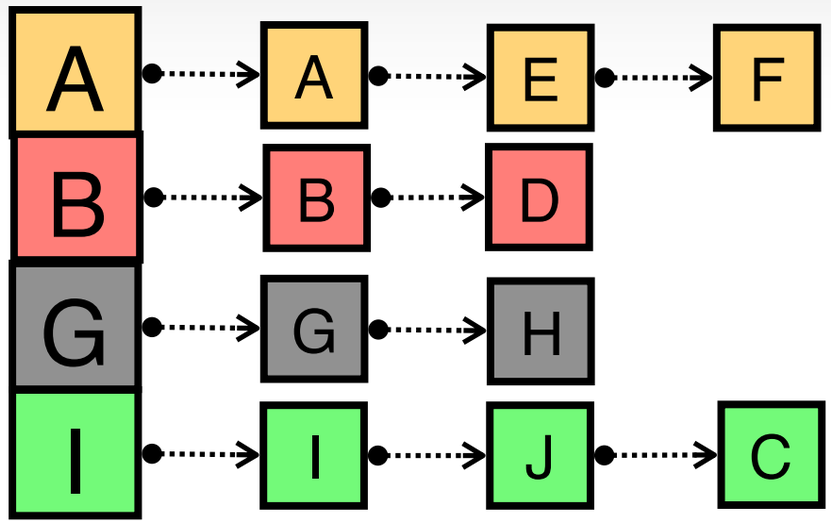


Kruskal MST

	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



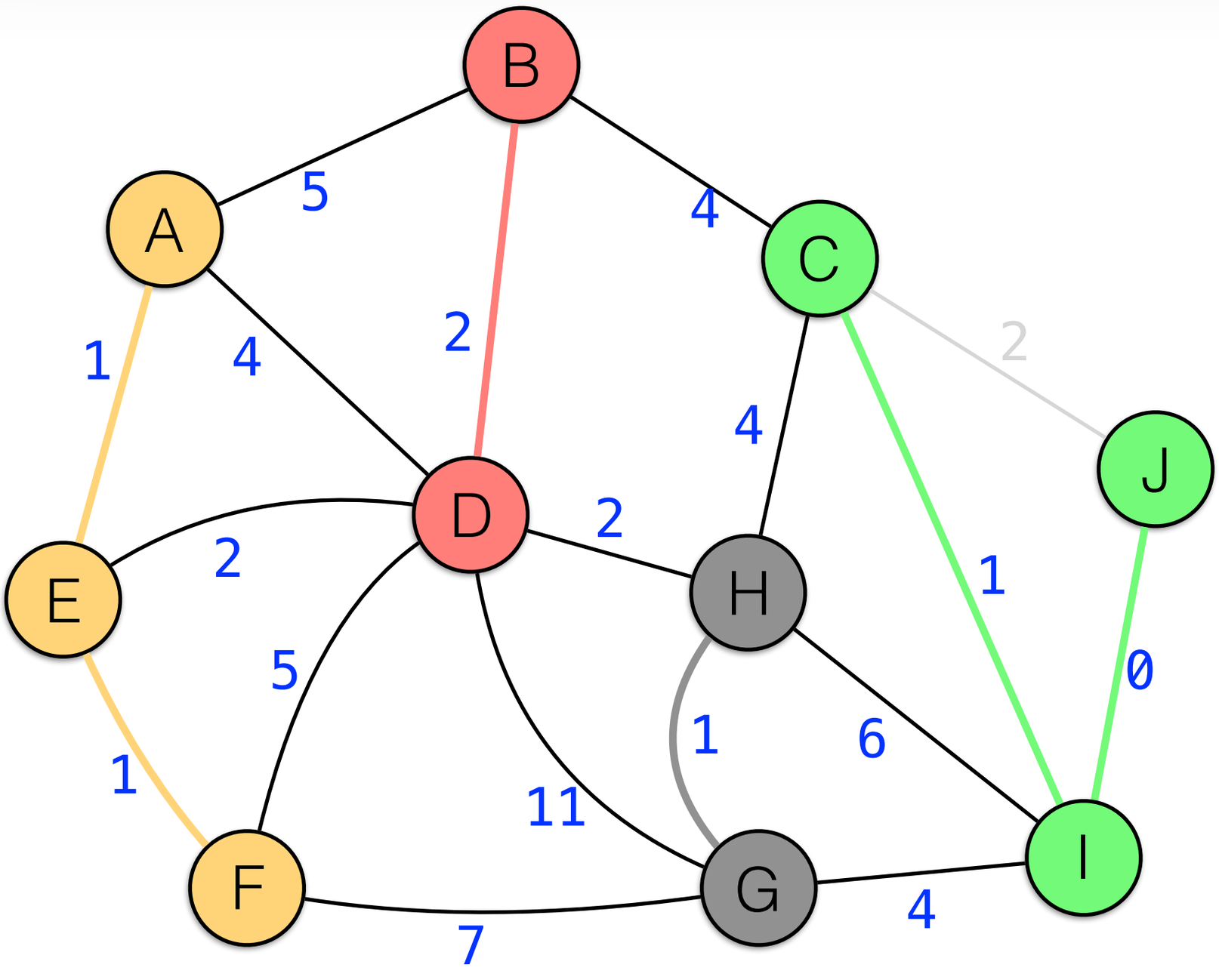

```
find(C)=I
find(J)=I
```

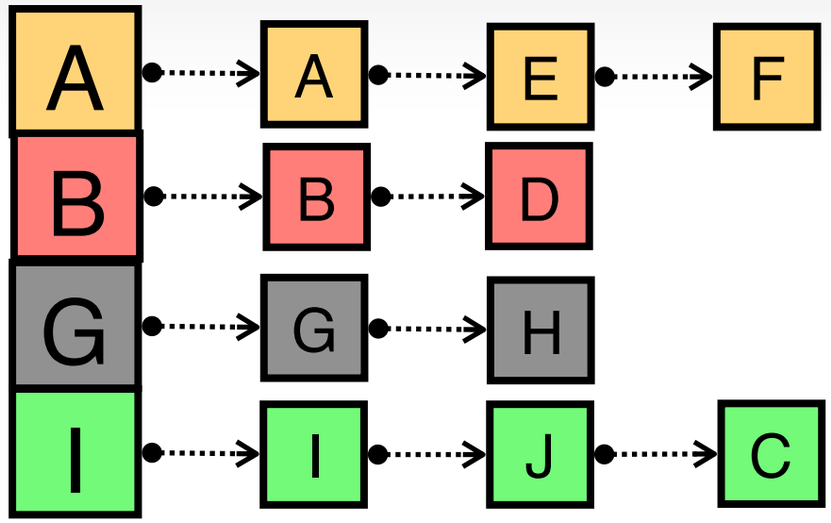


Nooooo, this generates a cycle!

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

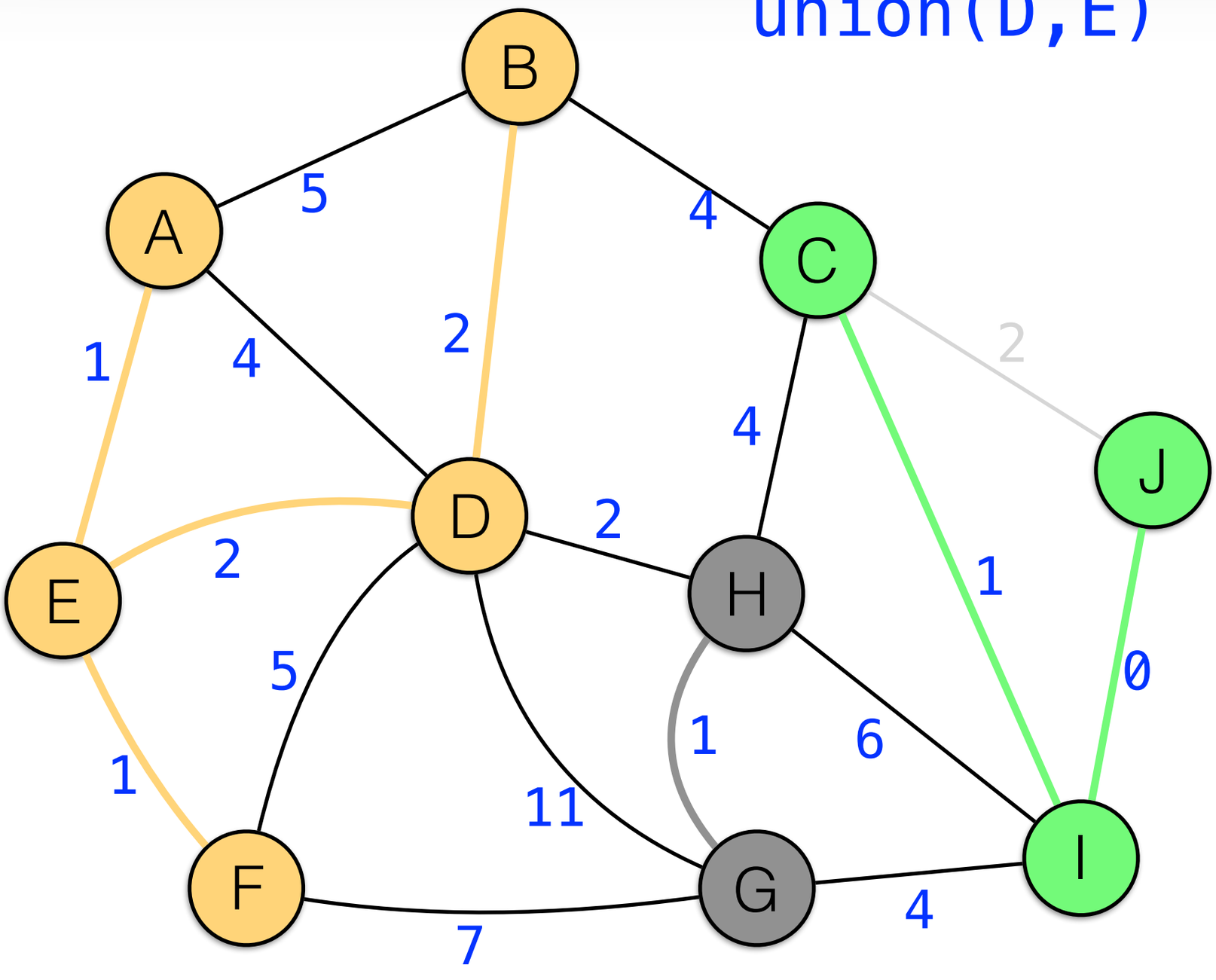


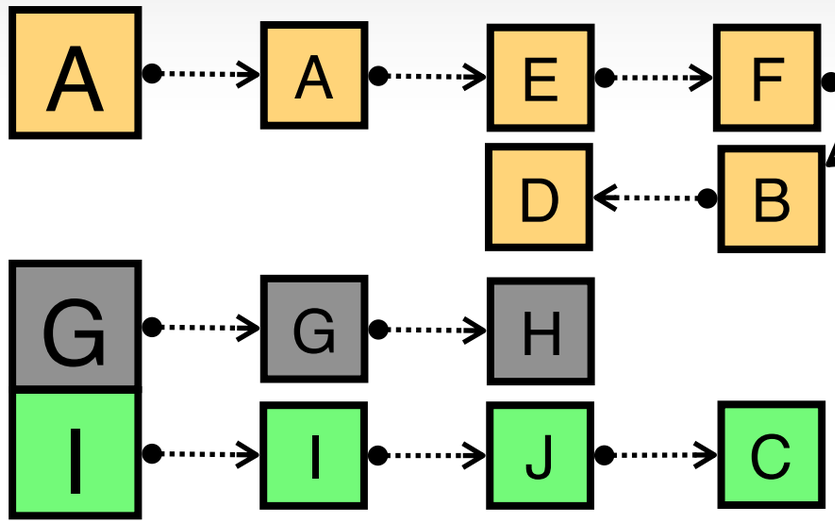


Kruskal MST

	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

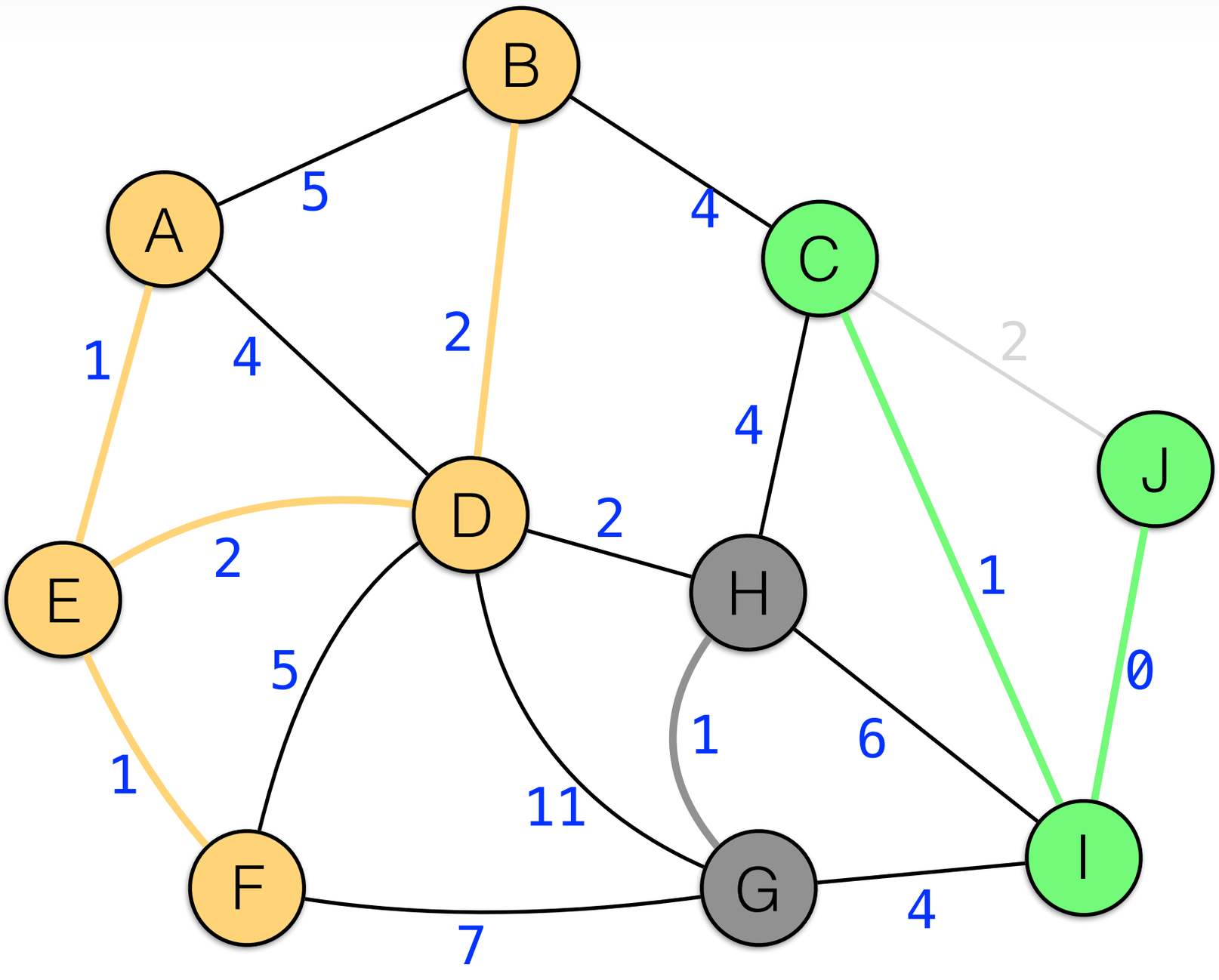
union(D,E)

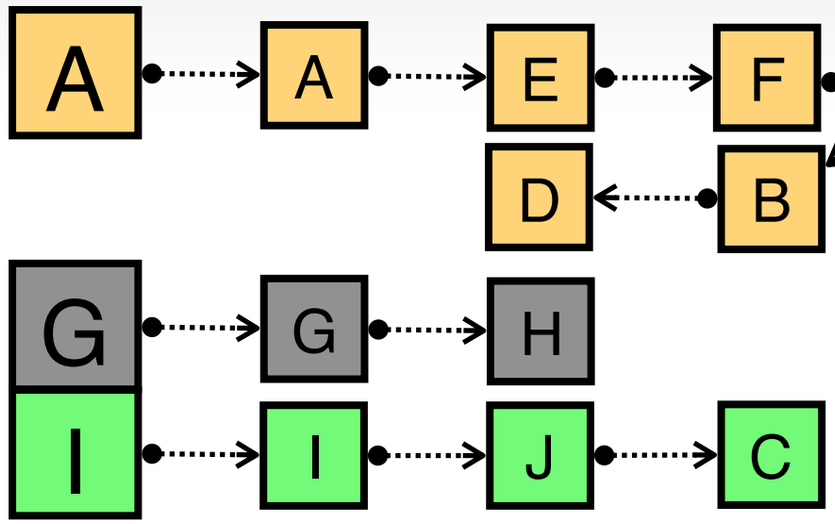




Kruskal MST

	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

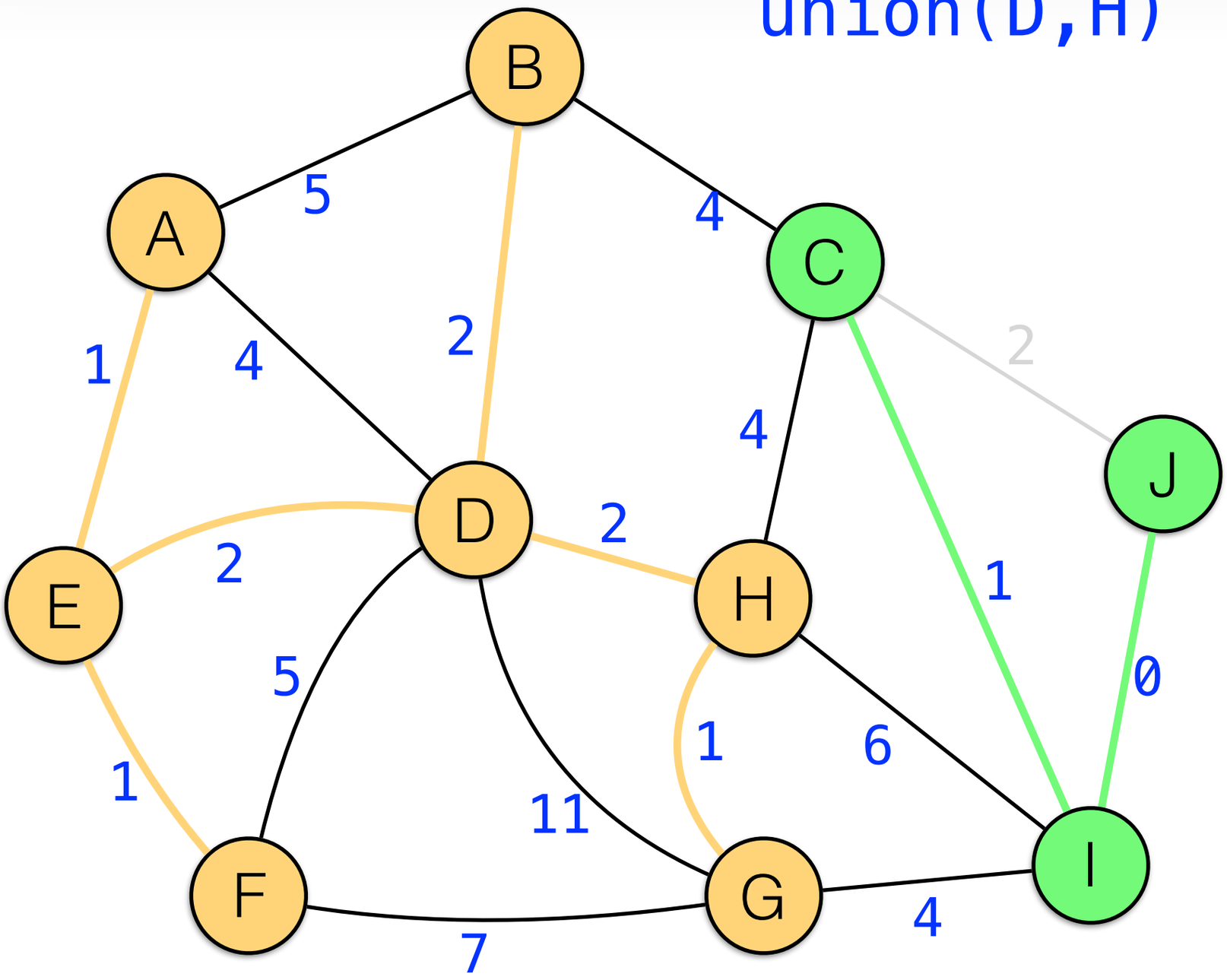


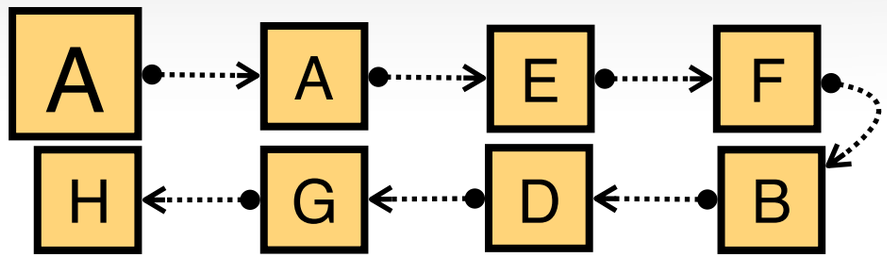


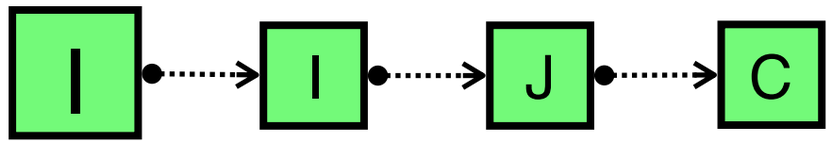
Kruskal MST

	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)

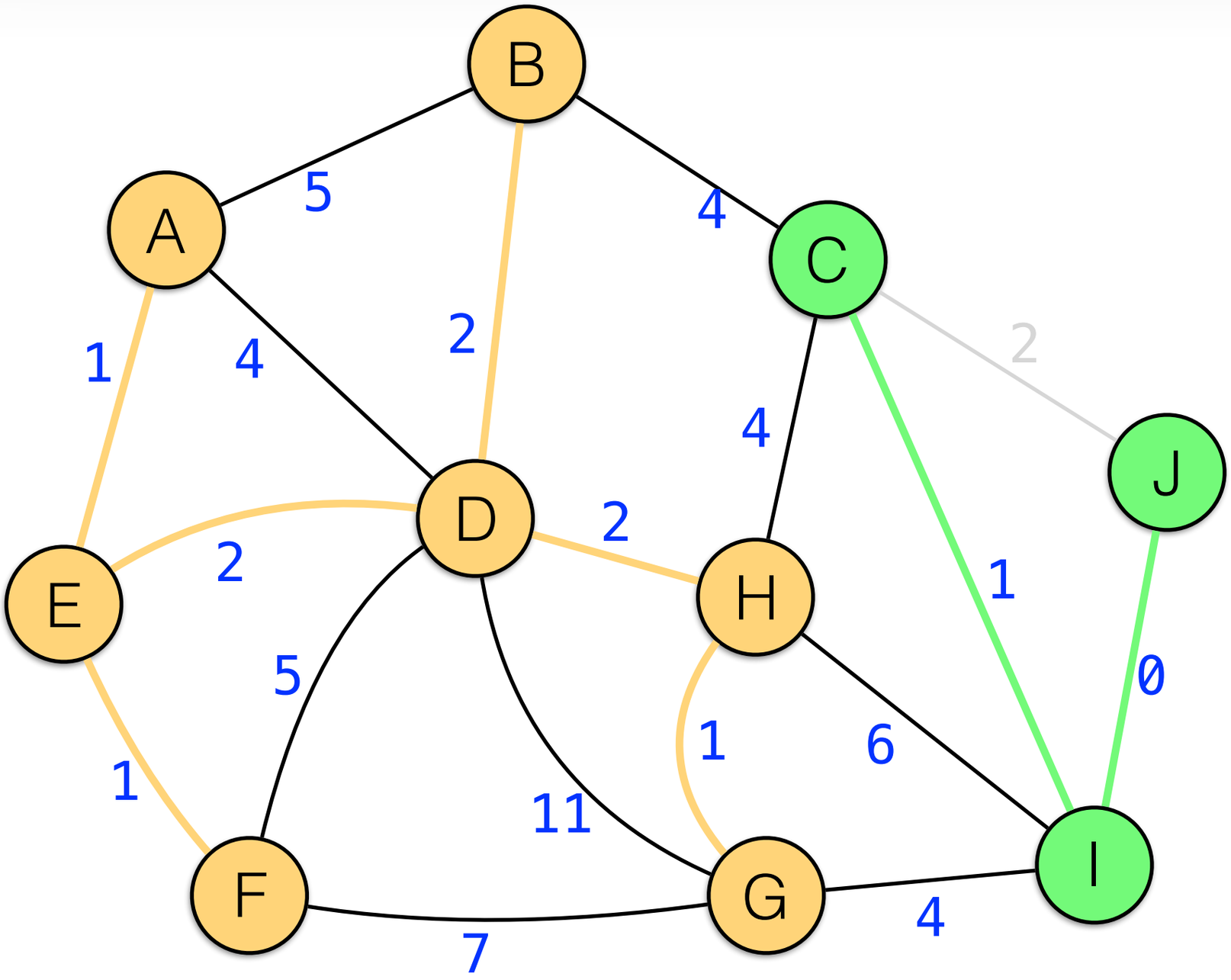


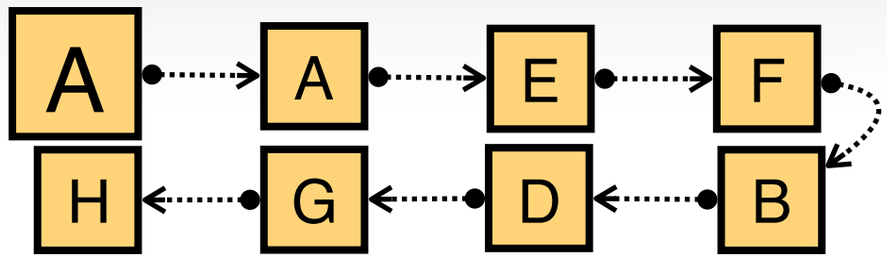


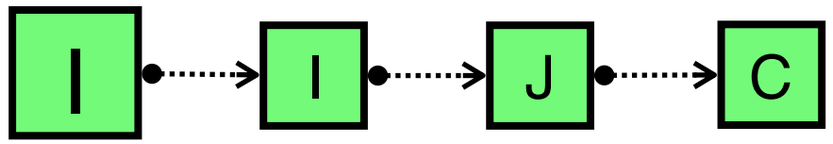


Kruskal MST

	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

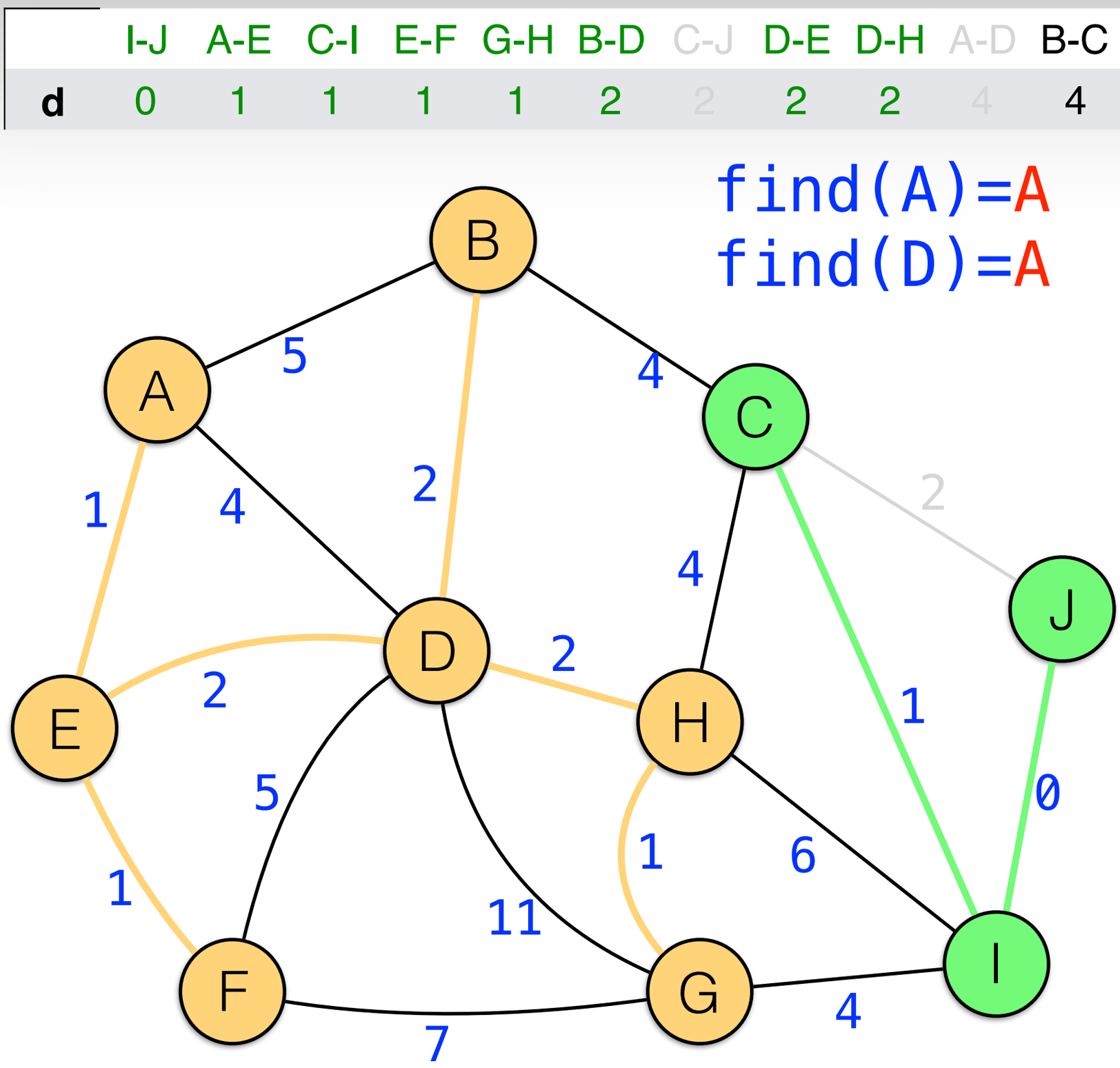




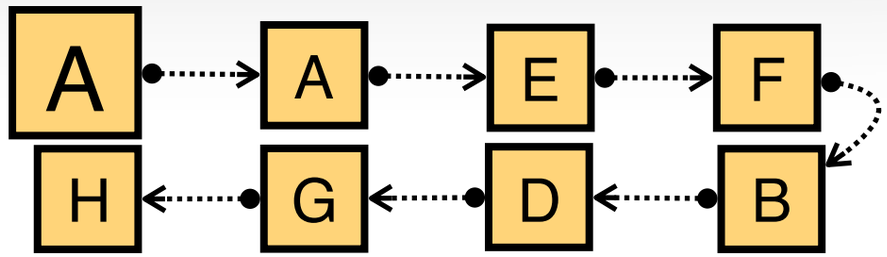


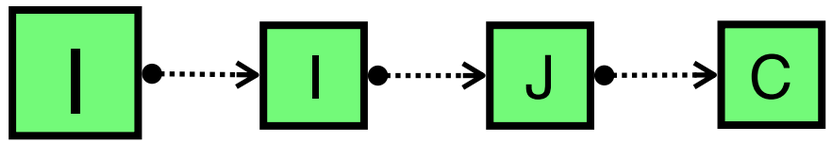
Kruskal MST

	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



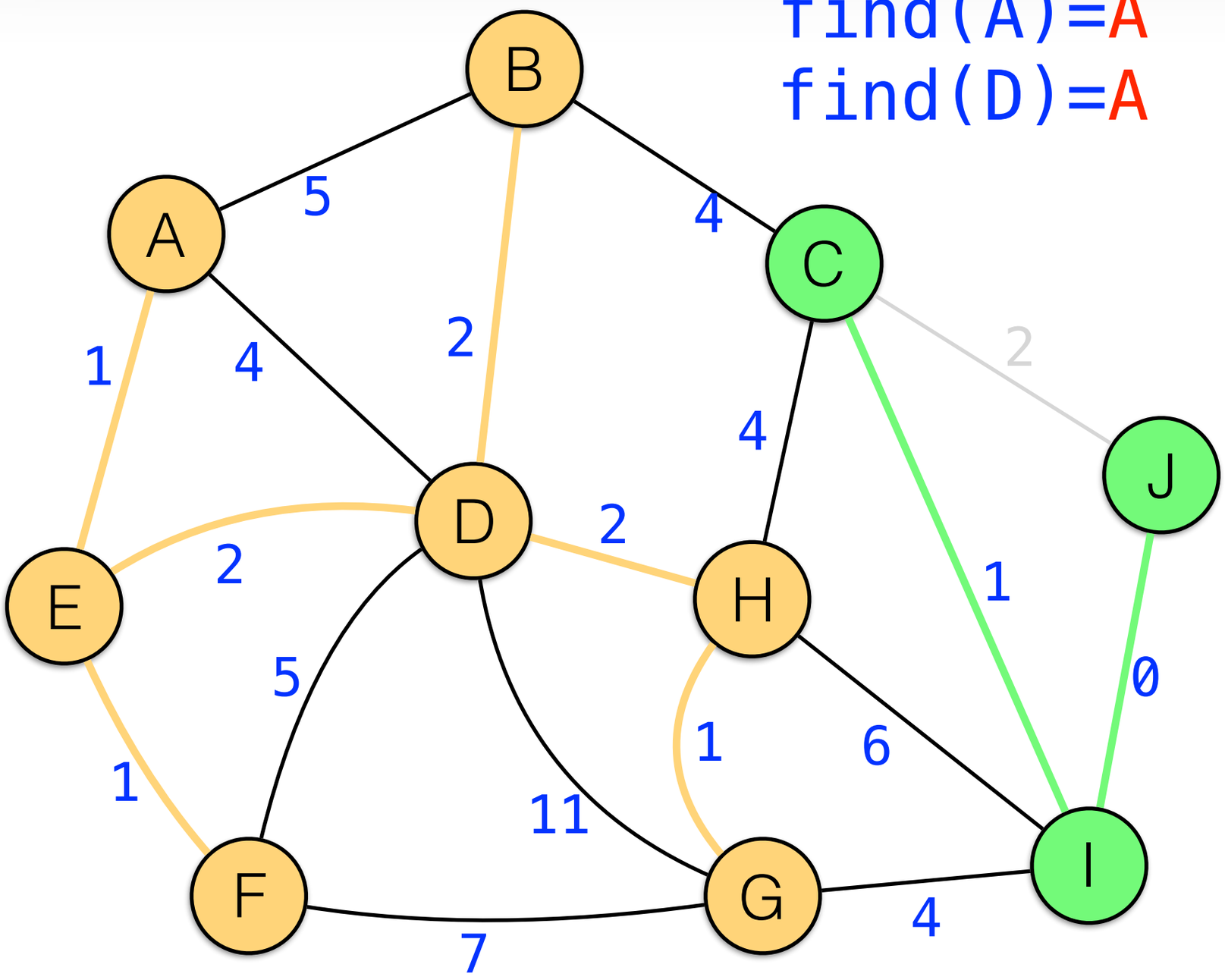

```
find(A)=A
find(D)=A
```



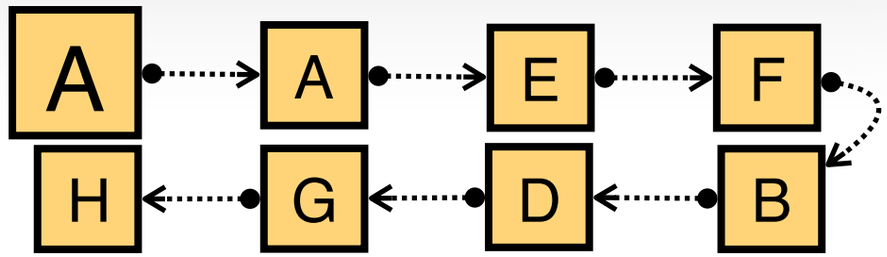


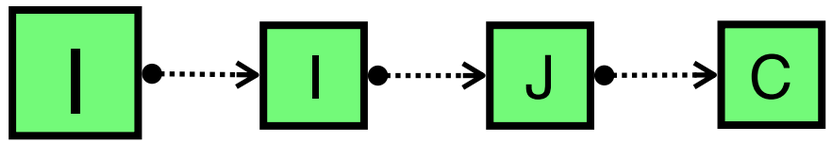
Kruskal MST

	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

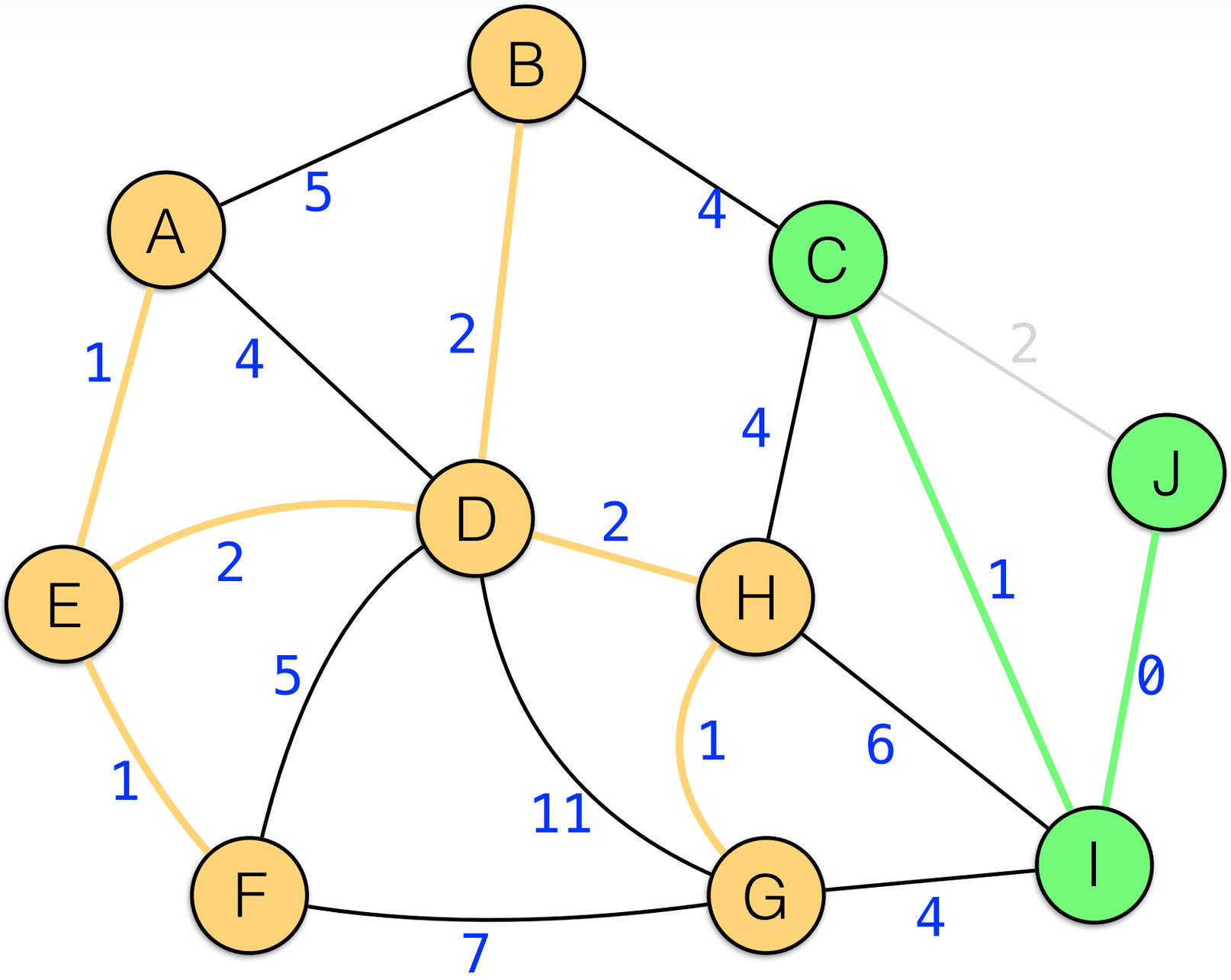


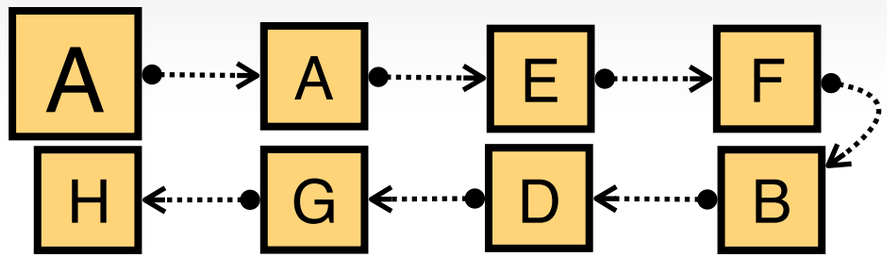


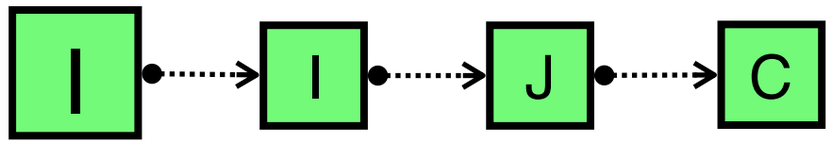
Do not add, it generates a cycle!

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

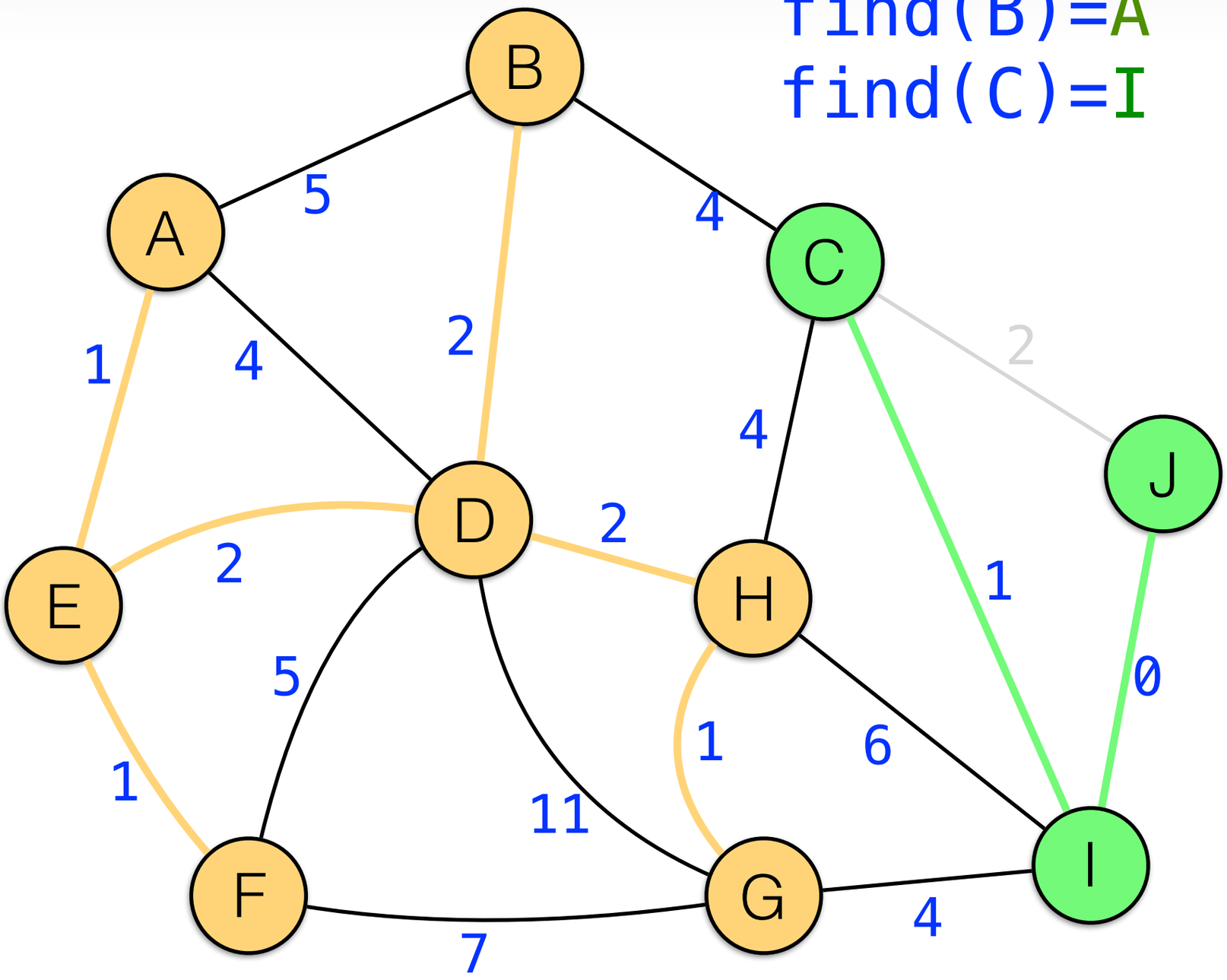




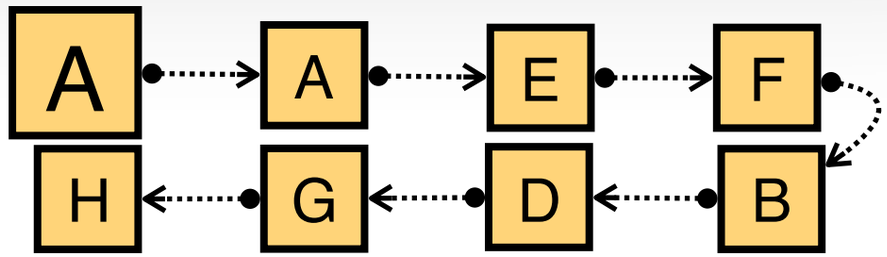


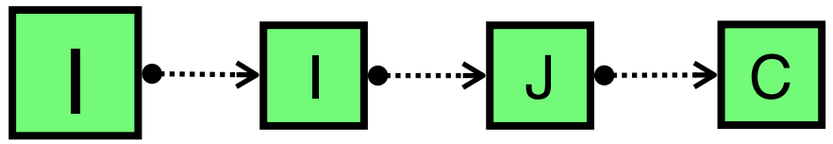
Kruskal MST

	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




```
find(B)=A
find(C)=I
```

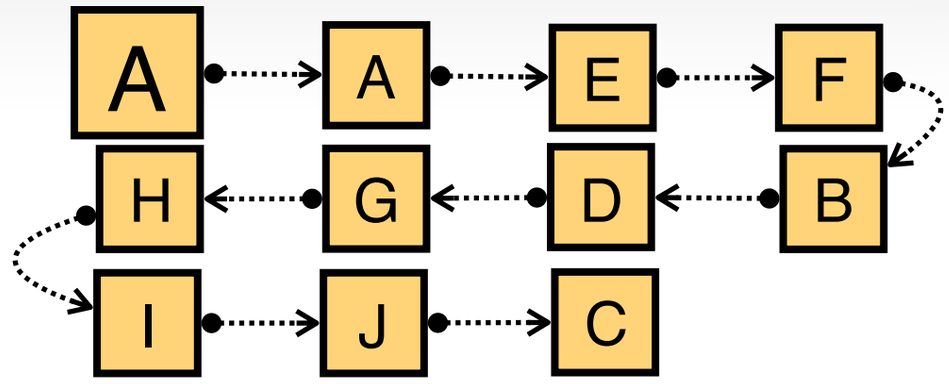


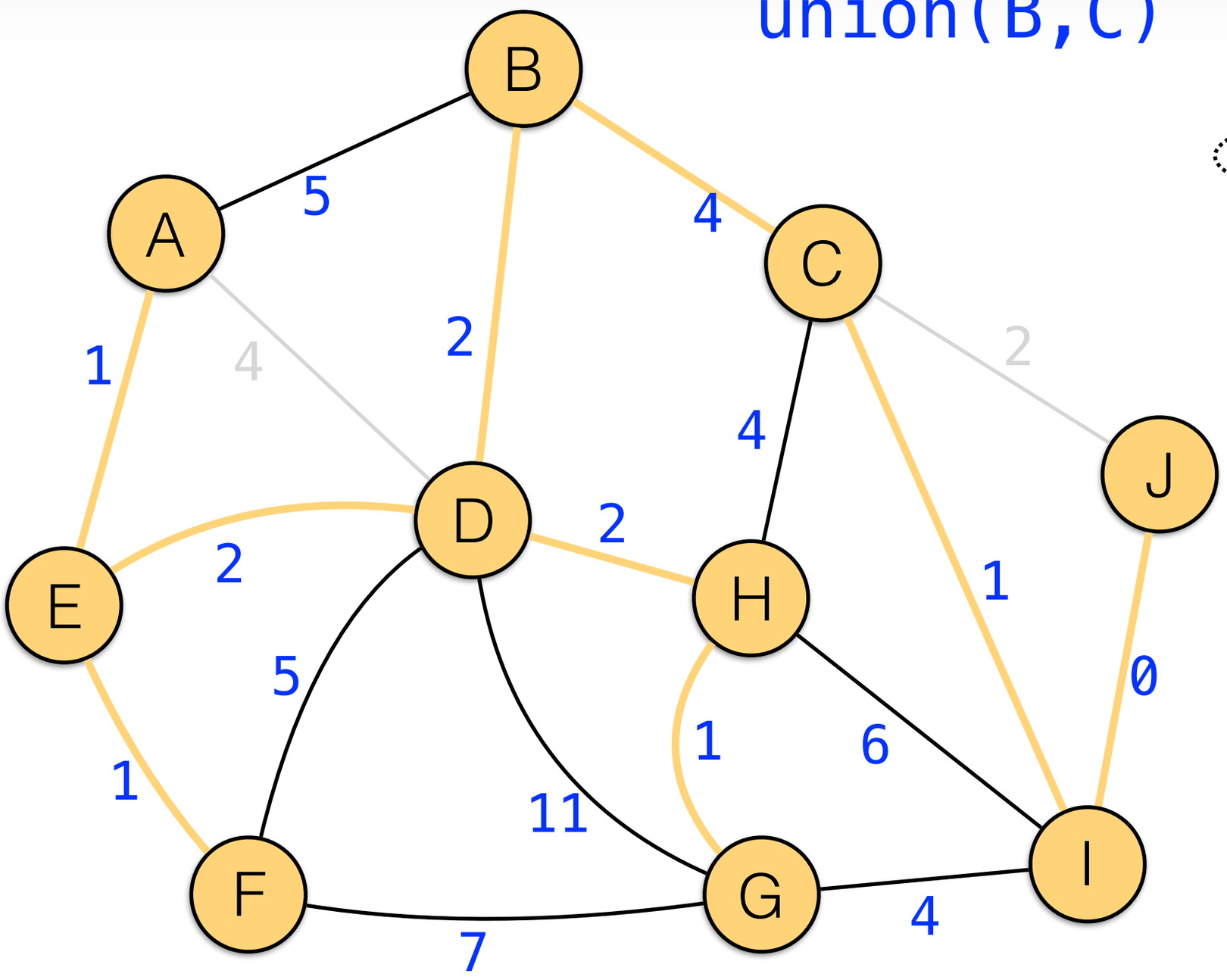


Kruskal MST

	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)

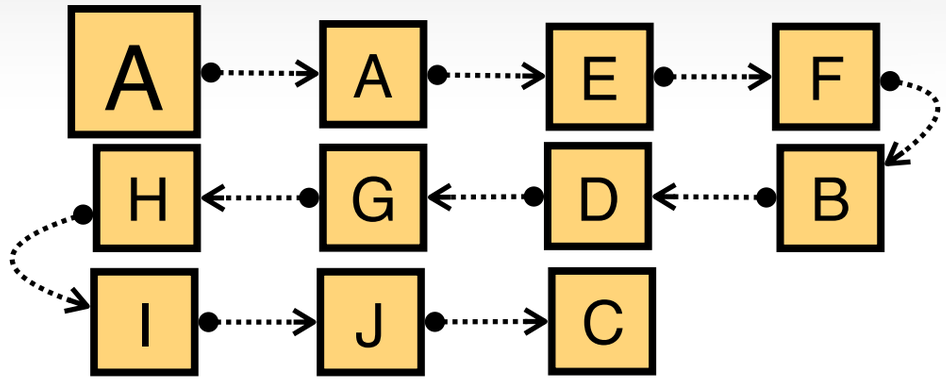


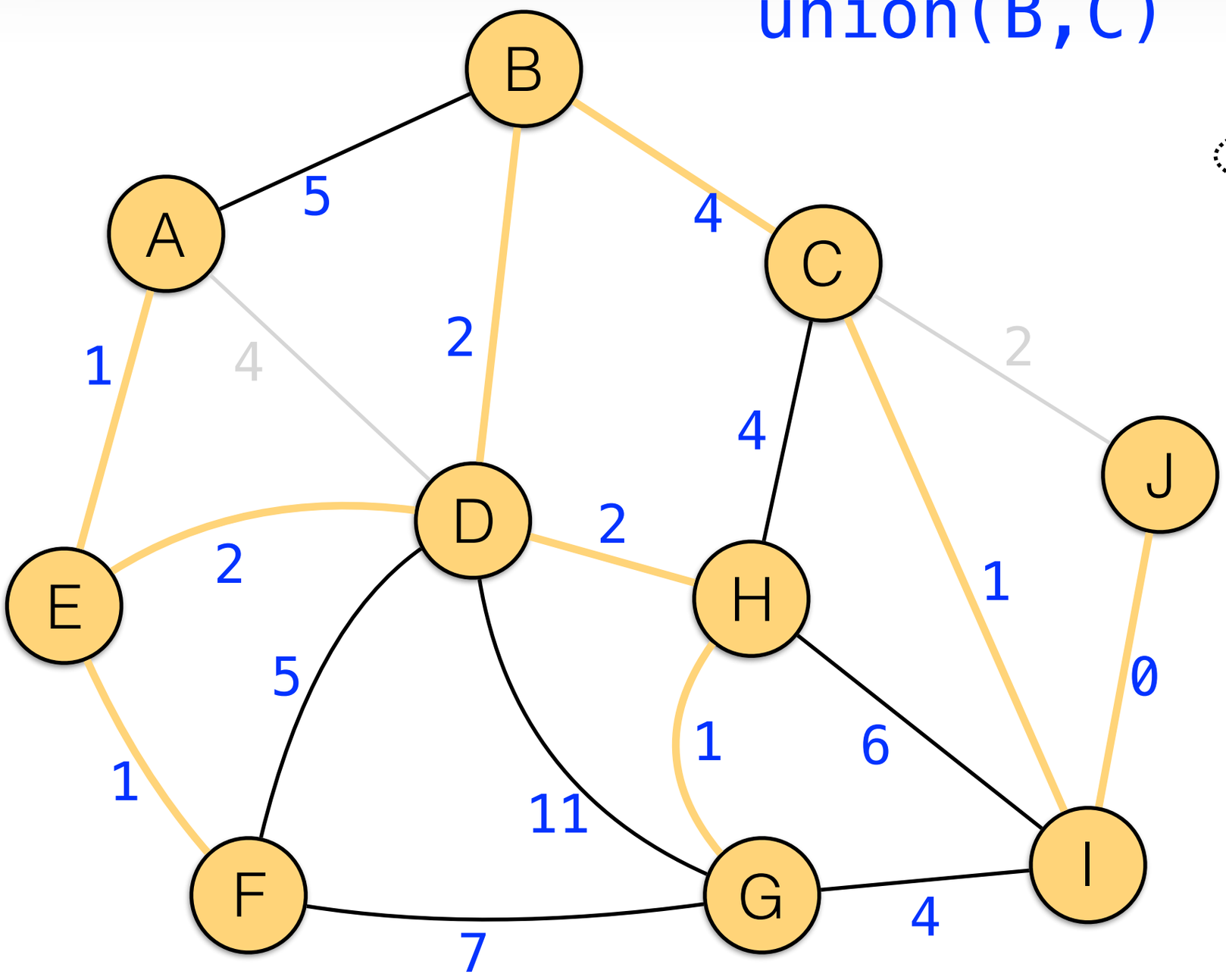


Kruskal MST

	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

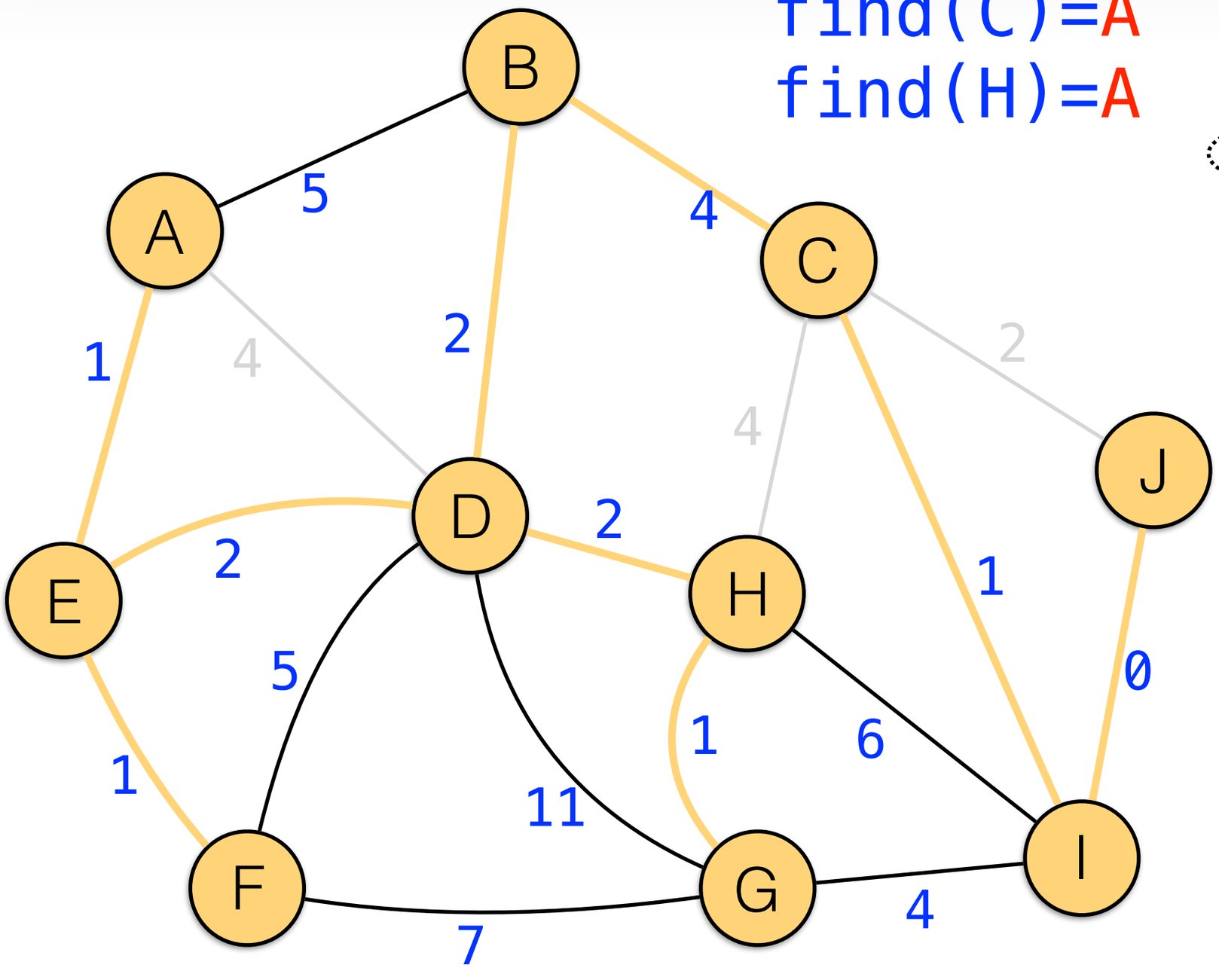
union(B,C)



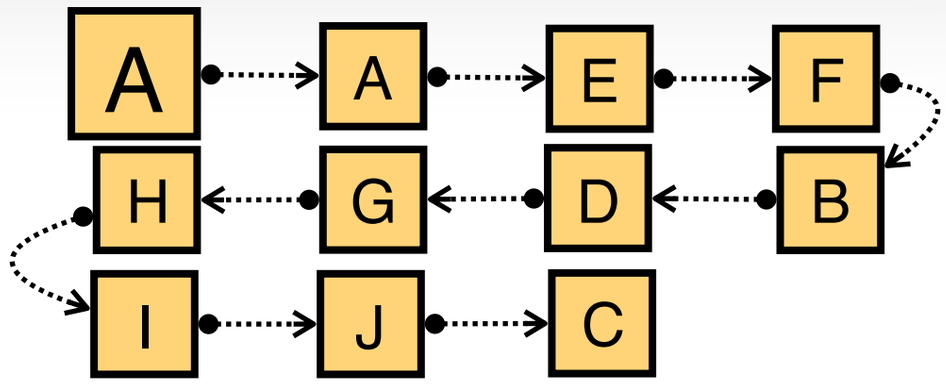


Kruskal MST

	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

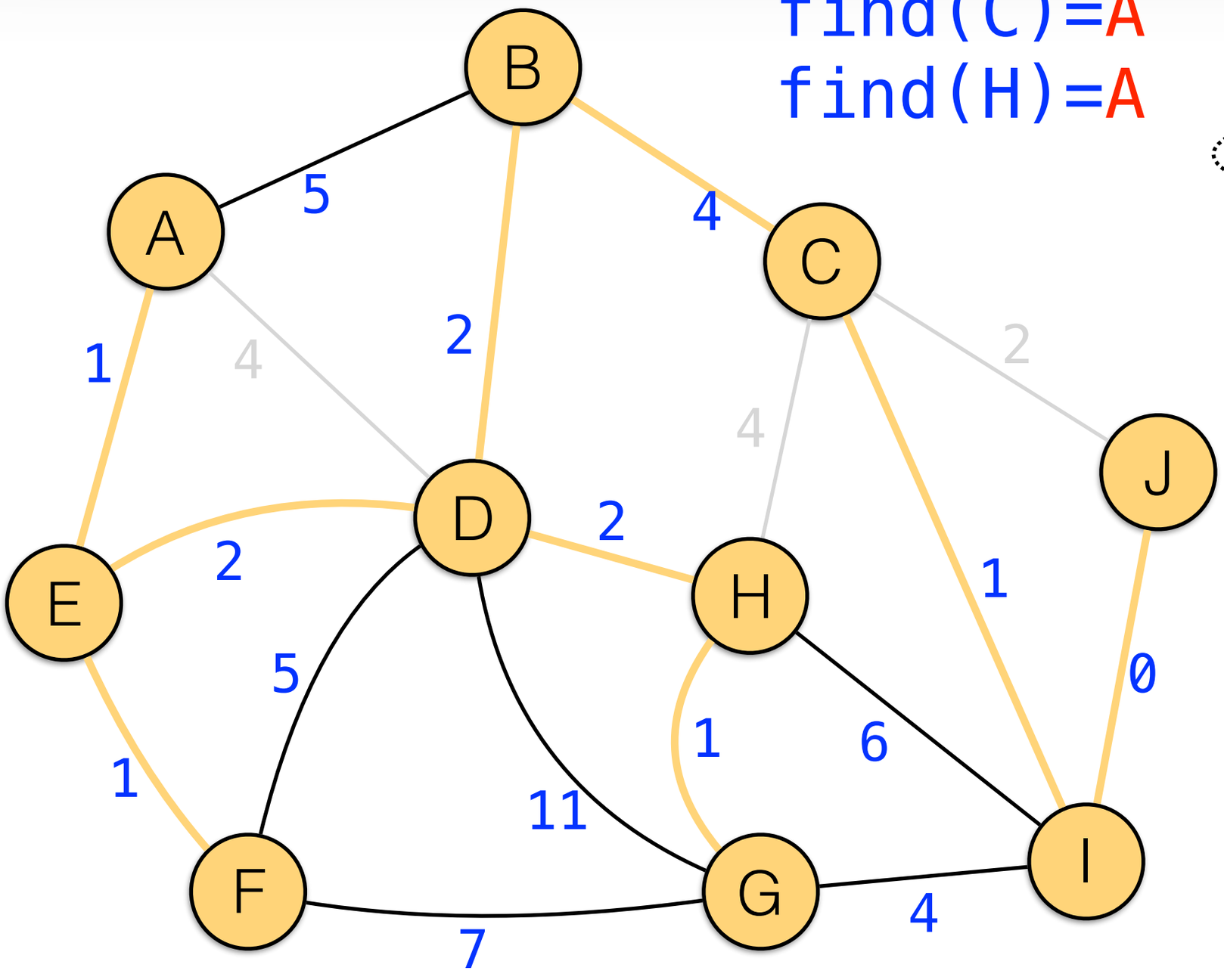



```
find(C)=A
find(H)=A
```

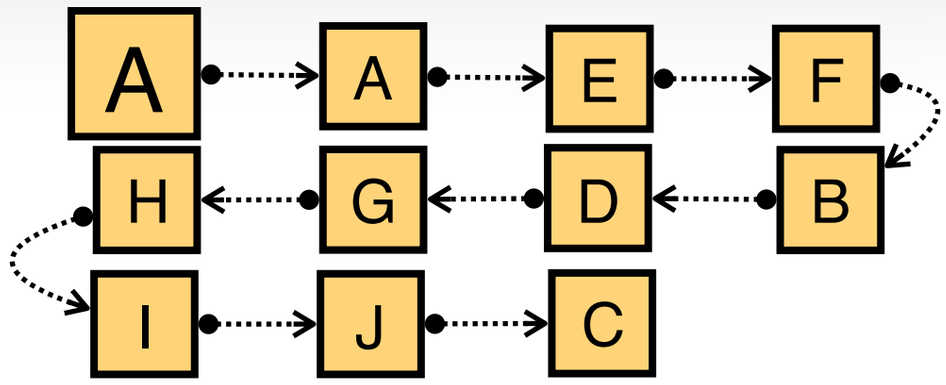


Kruskal MST

	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



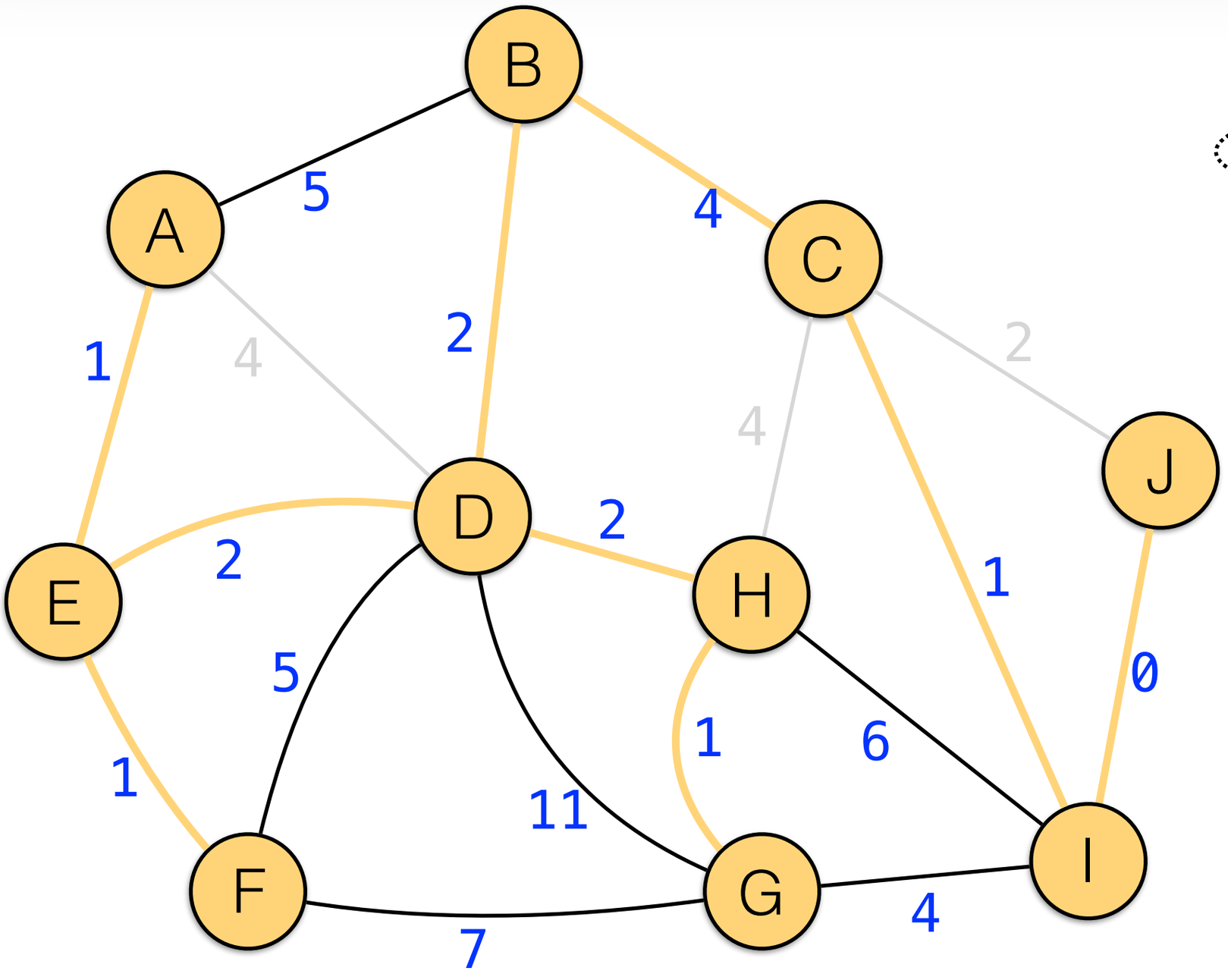

```
find(C)=A
find(H)=A
```

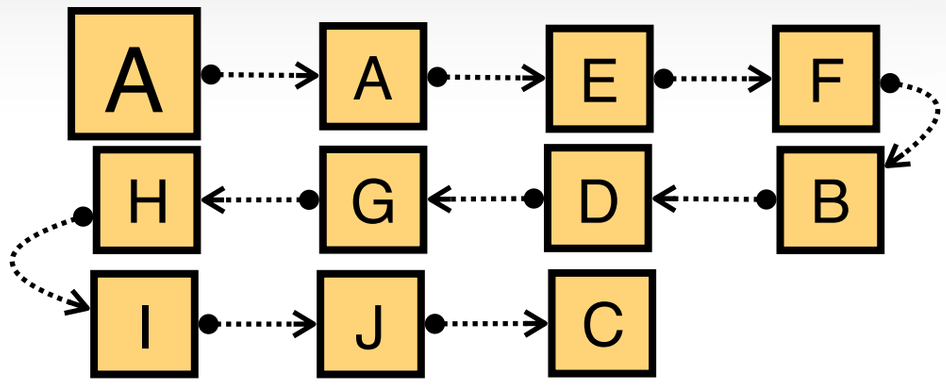


Do not add, it generates a cycle!

Kruskal MST

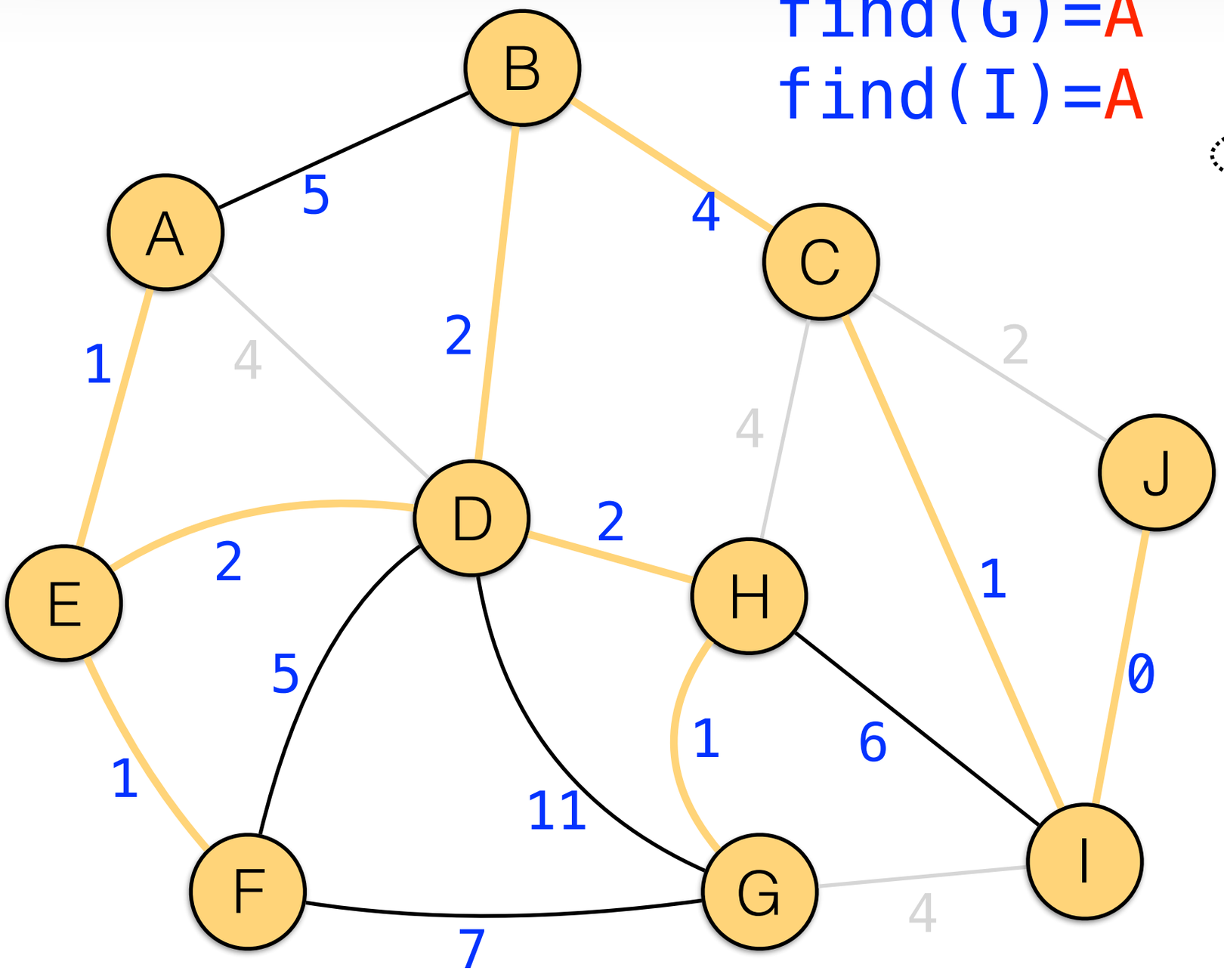
	I-J	A-E	C-I	E-F	G-H	B-D	C-J	D-E	D-H	A-D	B-C	C-H	G-I	A-B	D-F	H-I	F-G	D-G
d	0	1	1	1	1	2	2	2	2	4	4	4	4	5	5	6	7	11



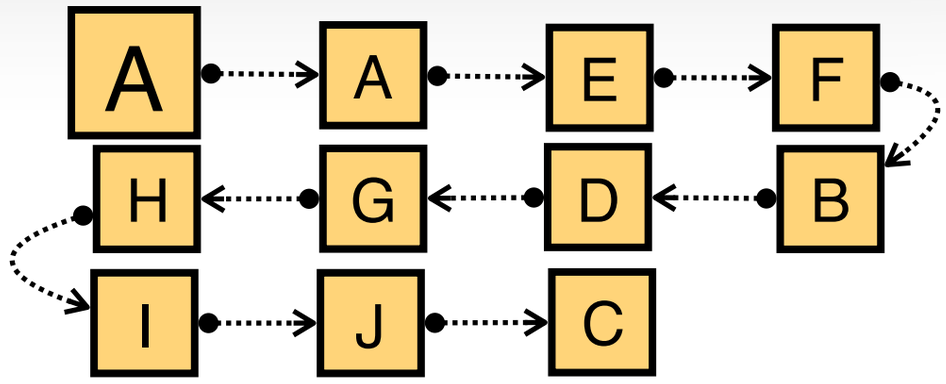


Kruskal MST

	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

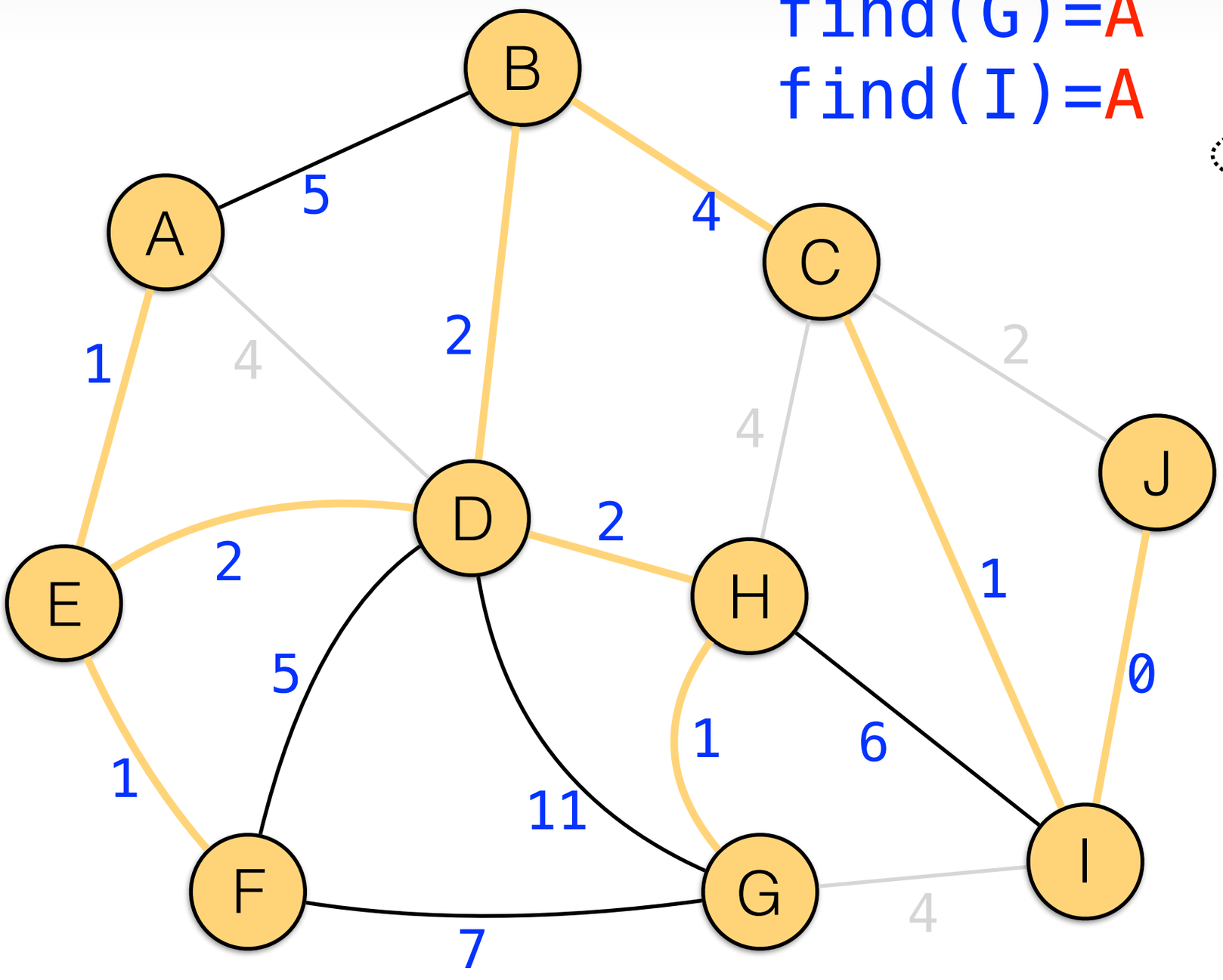



```
find(G)=A
find(I)=A
```

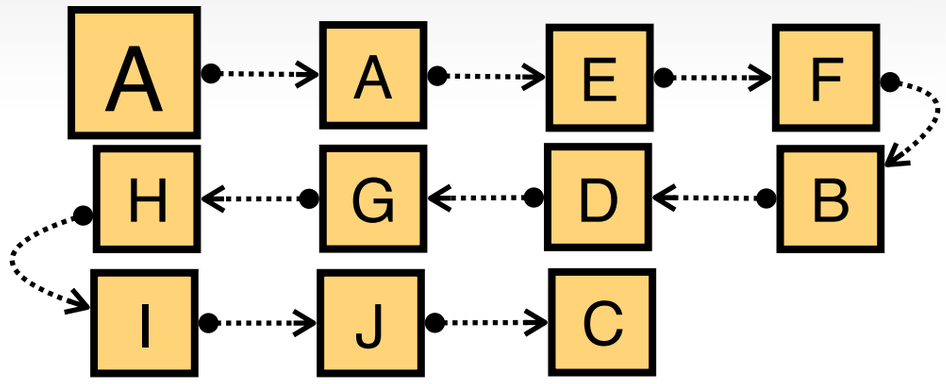


Kruskal MST

	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



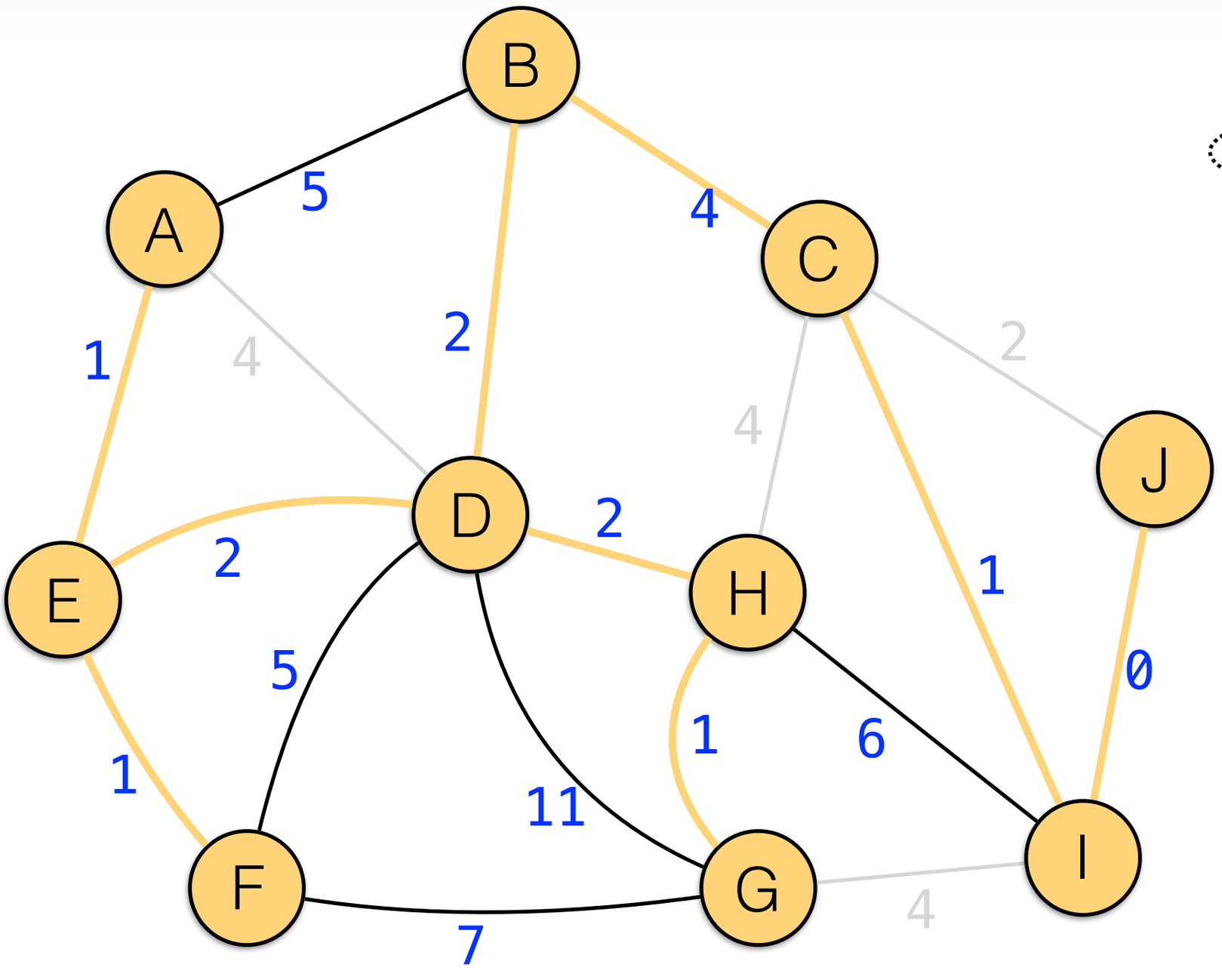
find(G)=A
find(I)=A

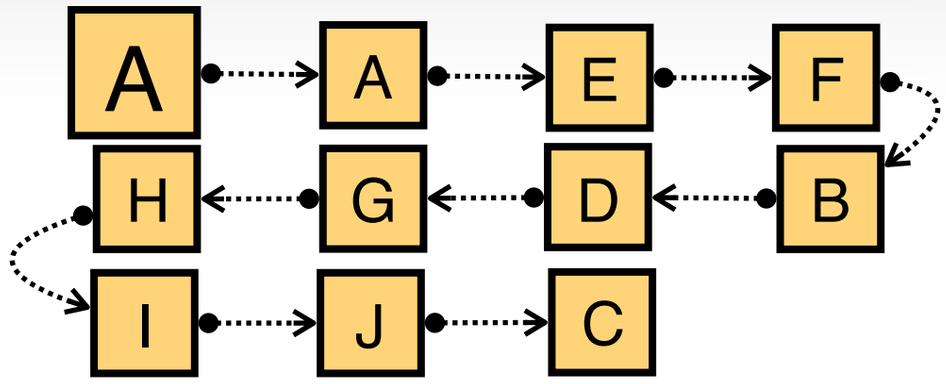


Do not add, it generates a cycle!

Kruskal MST

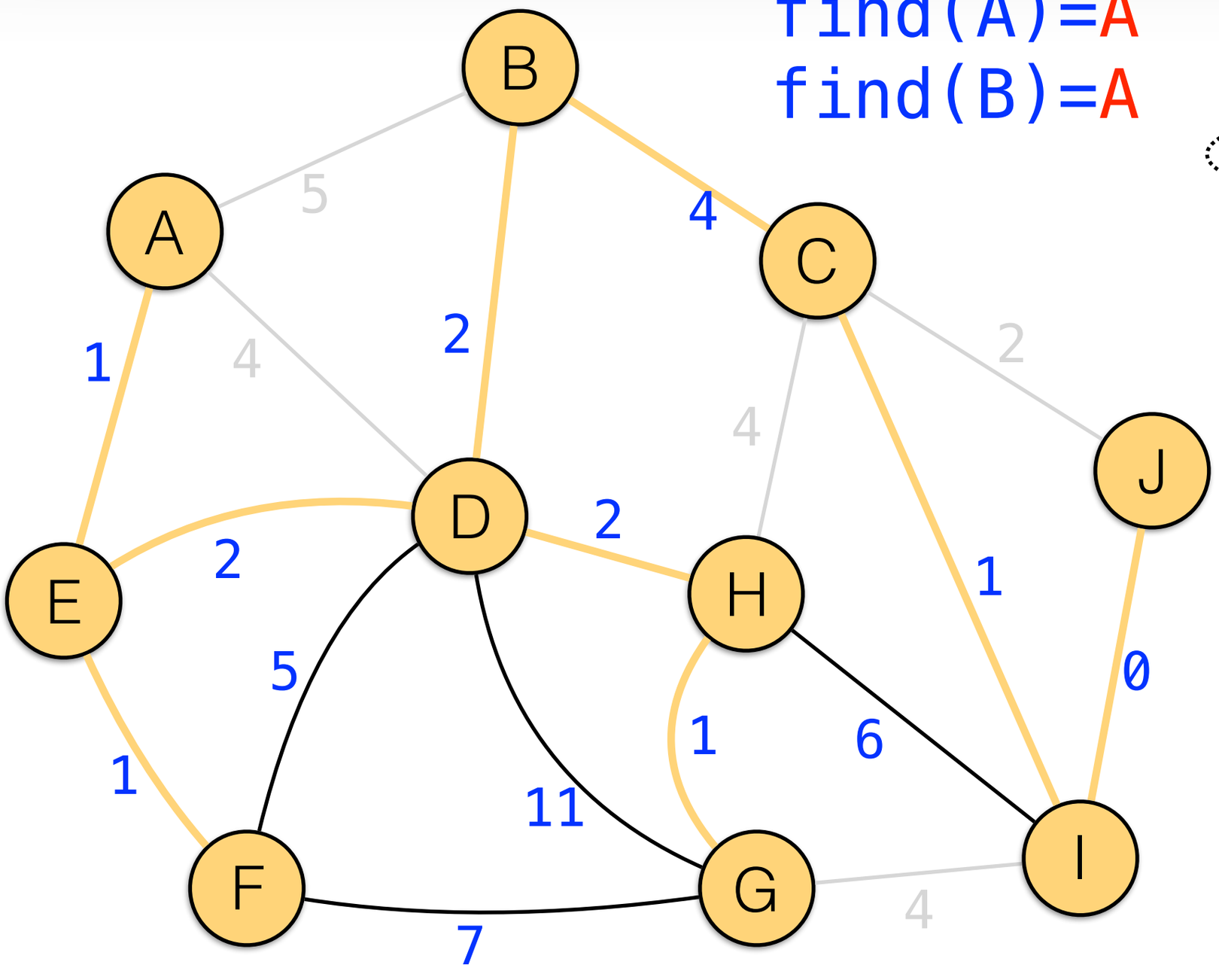
	I-J	A-E	C-I	E-F	G-H	B-D	C-J	D-E	D-H	A-D	B-C	C-H	G-I	A-B	D-F	H-I	F-G	D-G
d	0	1	1	1	1	2	2	2	2	4	4	4	4	5	5	6	7	11



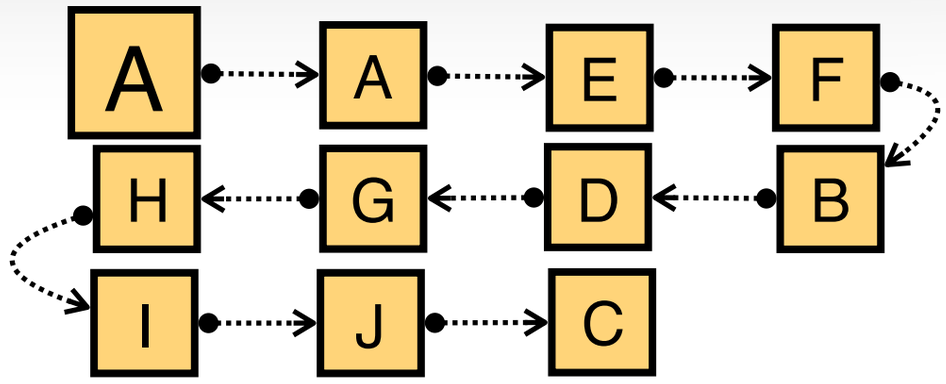


Kruskal MST

	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

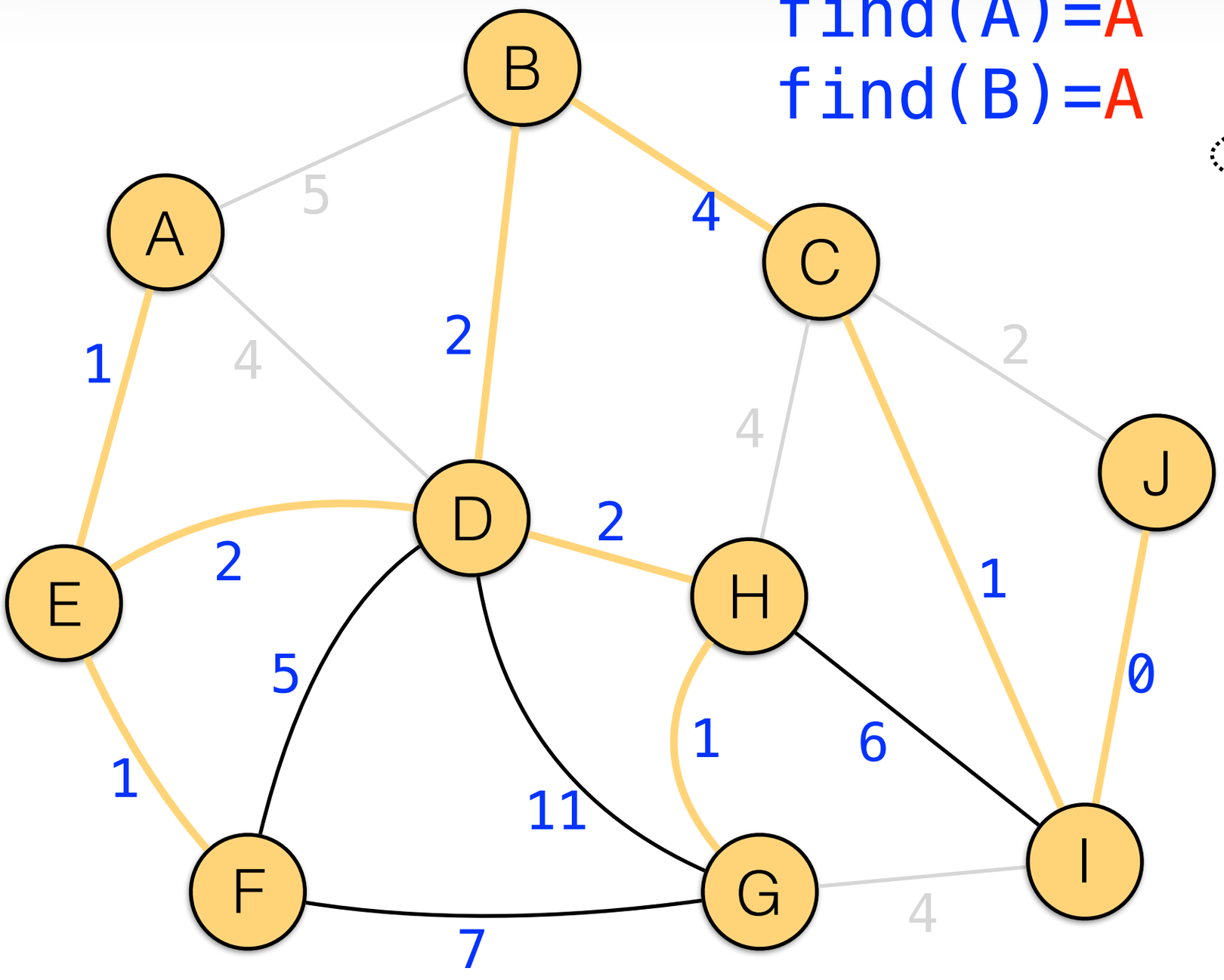



```
find(A)=A
find(B)=A
```

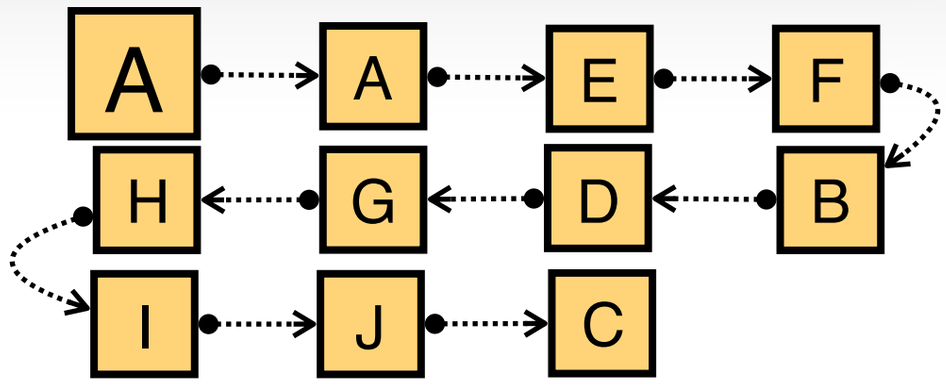


Kruskal MST

	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



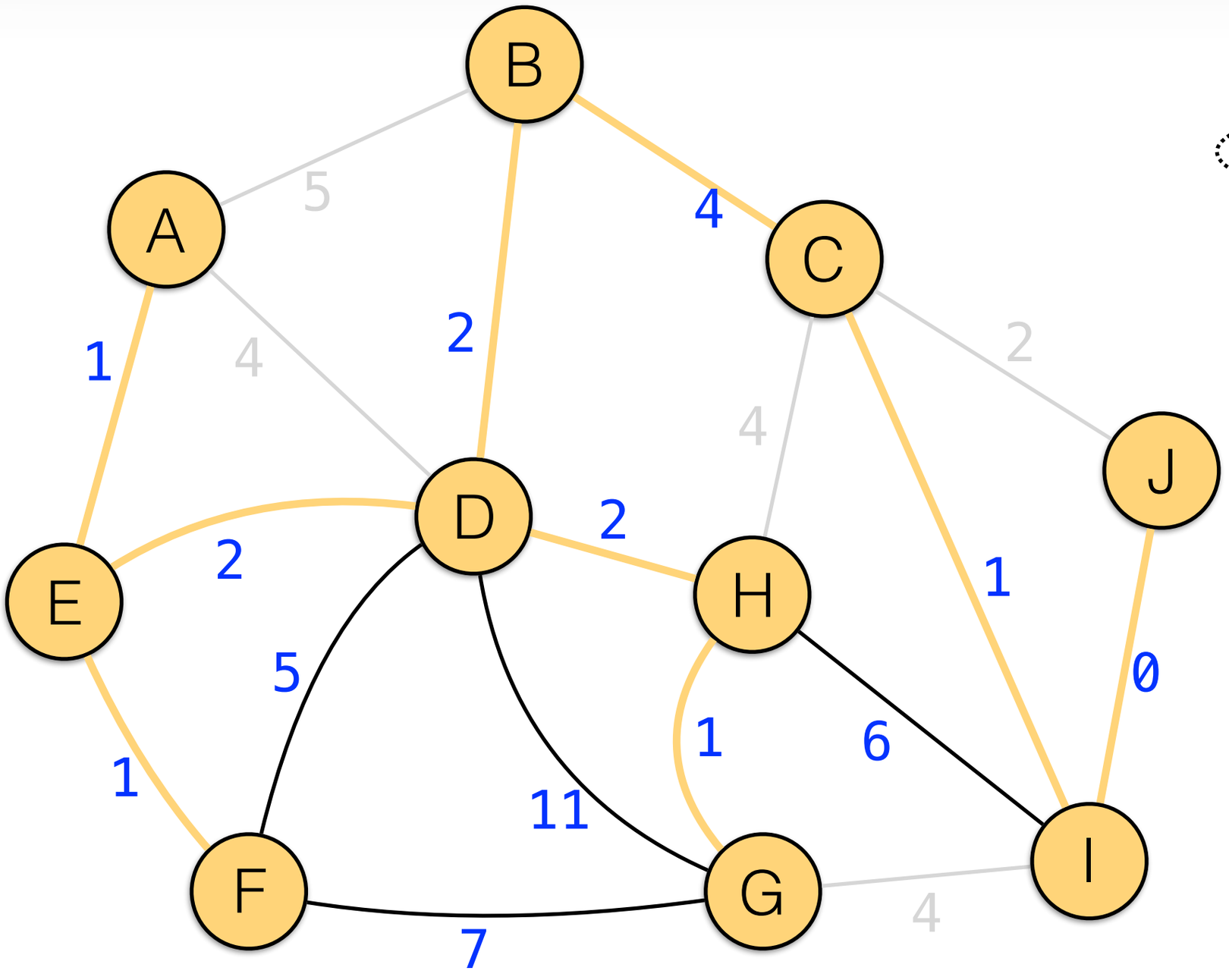

```
find(A)=A
find(B)=A
```

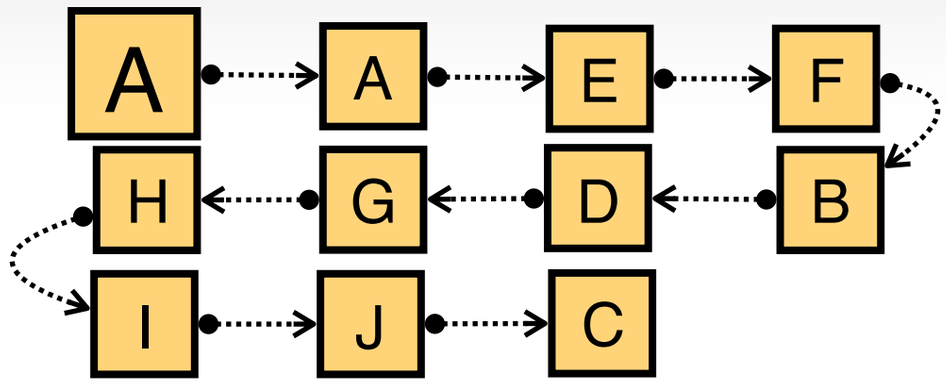


Do not add, it generates a cycle!

Kruskal MST

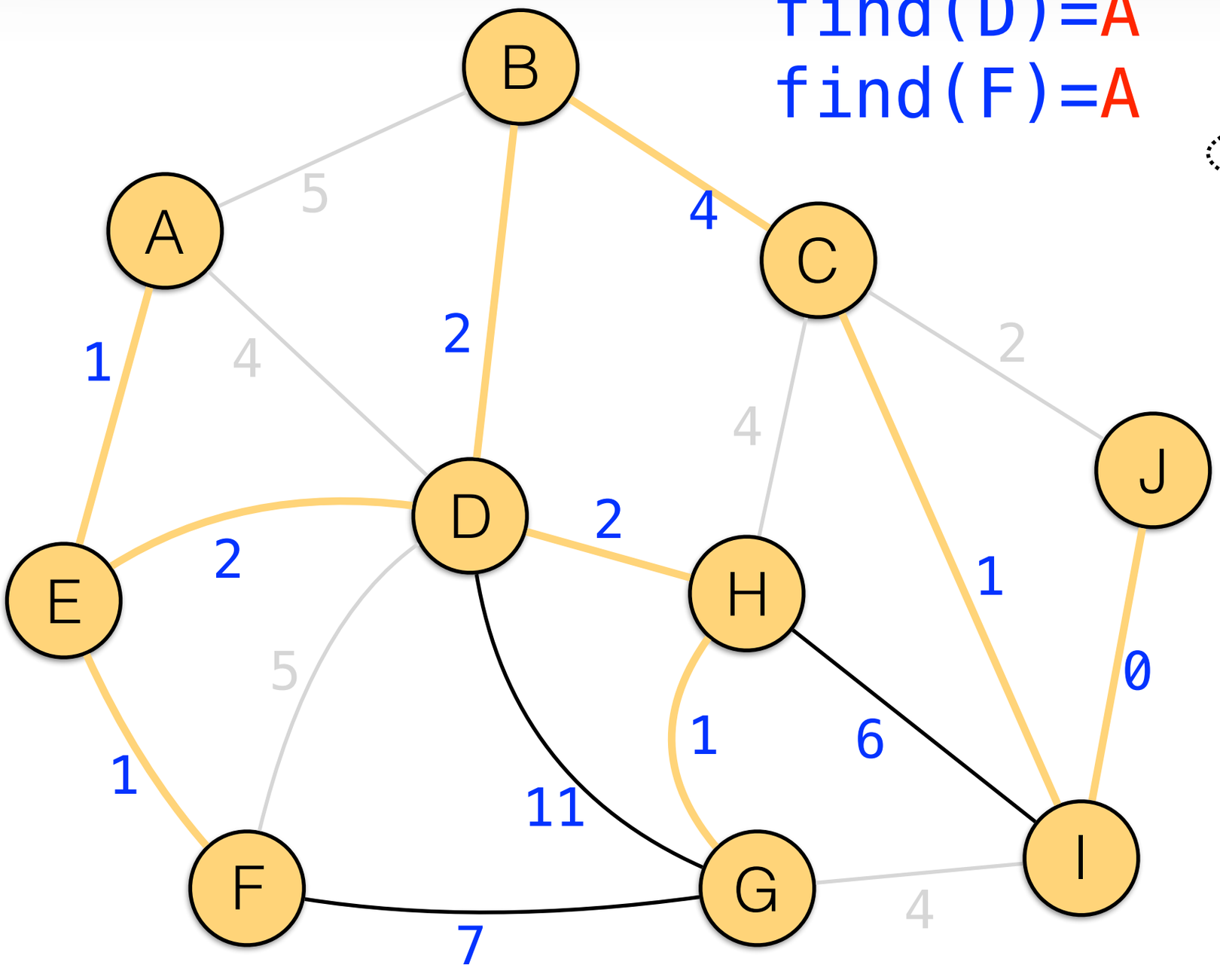
	I-J	A-E	C-I	E-F	G-H	B-D	C-J	D-E	D-H	A-D	B-C	C-H	G-I	A-B	D-F	H-I	F-G	D-G
d	0	1	1	1	1	2	2	2	2	4	4	4	4	5	5	6	7	11



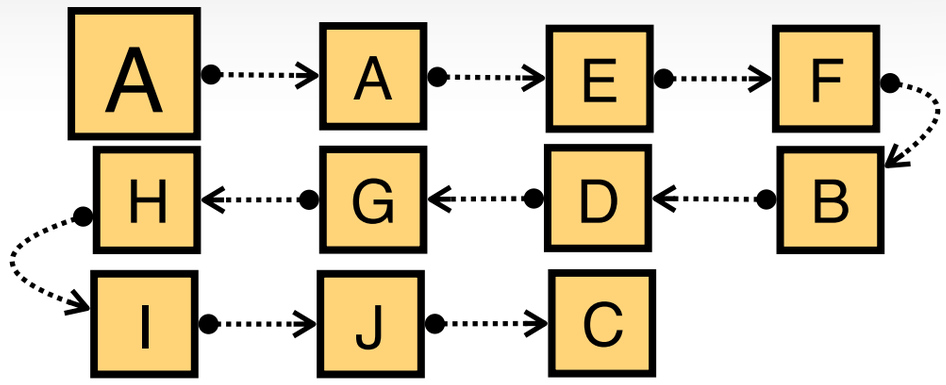


Kruskal MST

	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

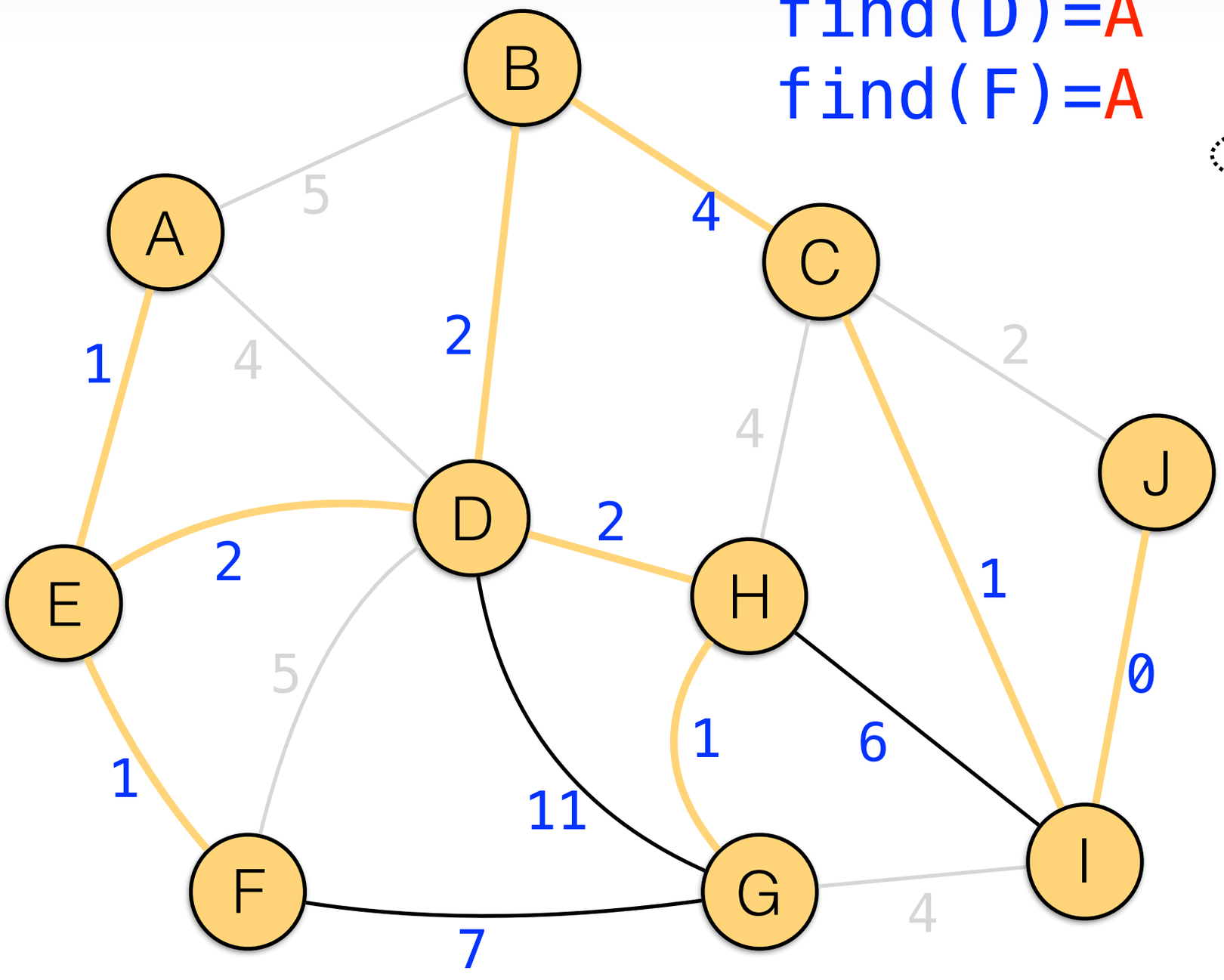



```
find(D)=A
find(F)=A
```

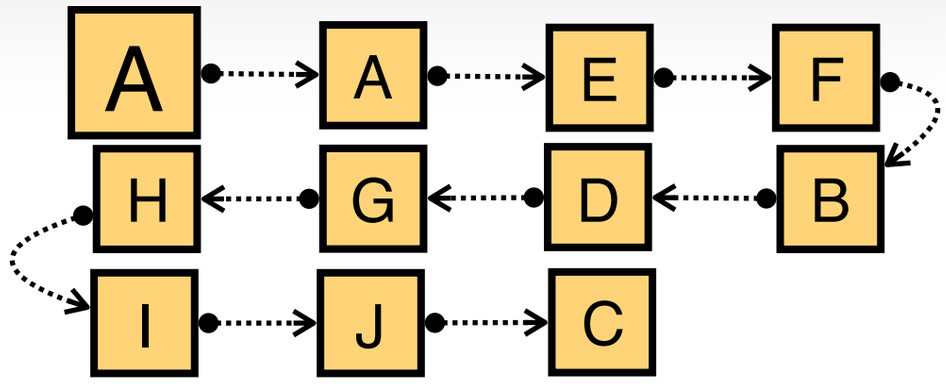


Kruskal MST

	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



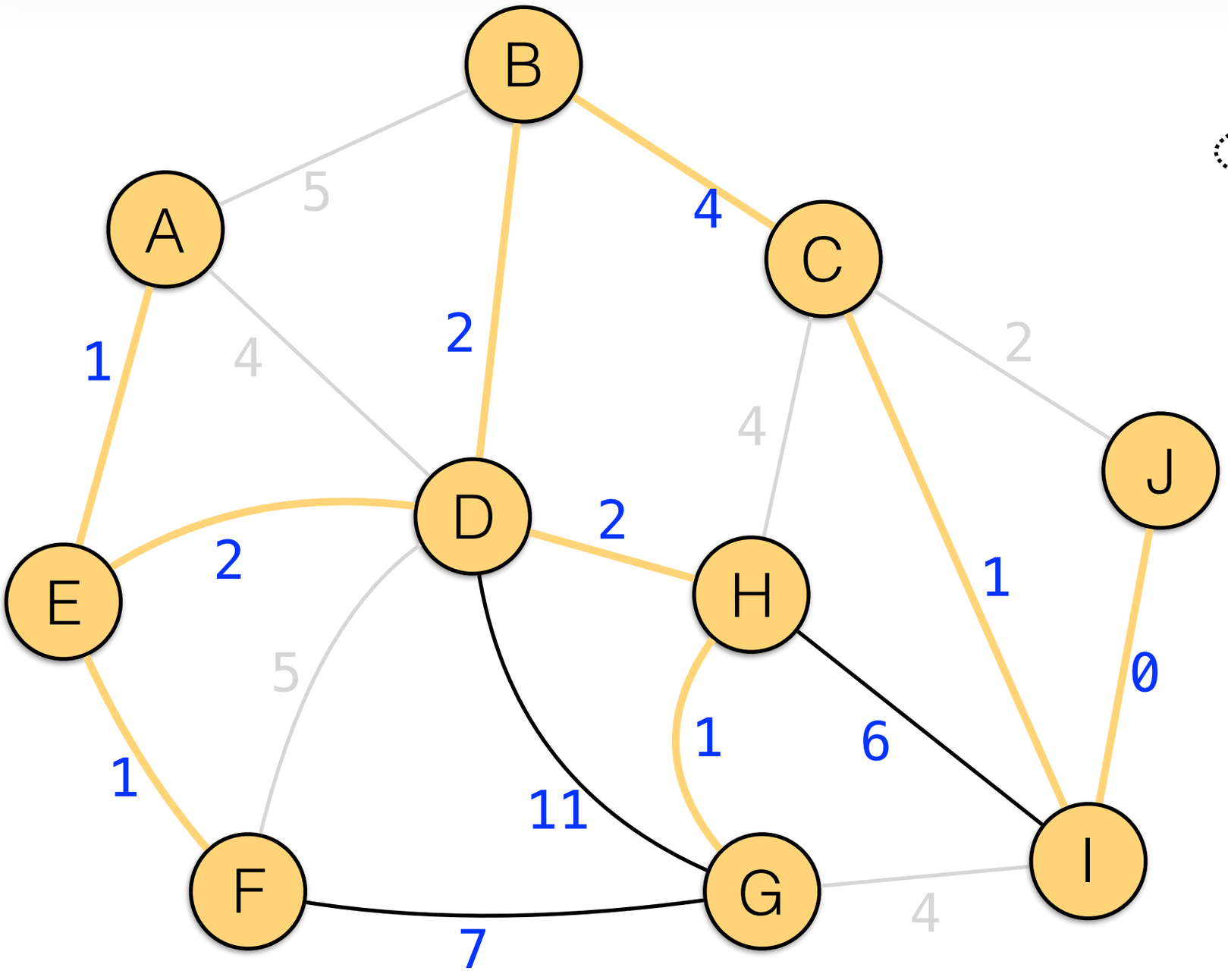

```
find(D)=A
find(F)=A
```

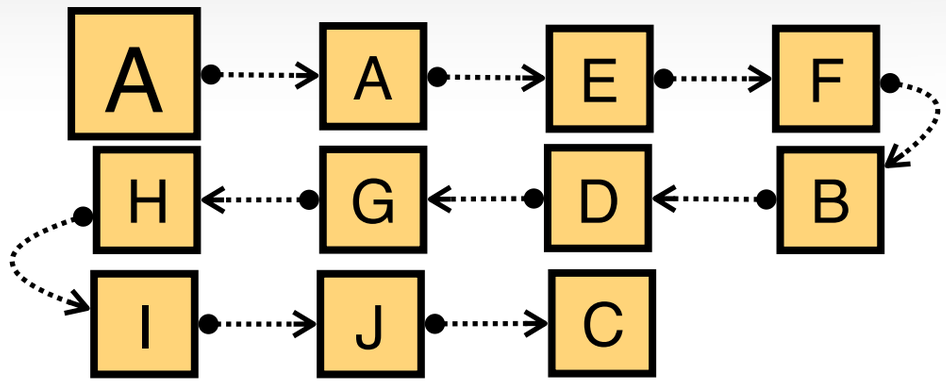


Do not add, it generates a cycle!

Kruskal MST

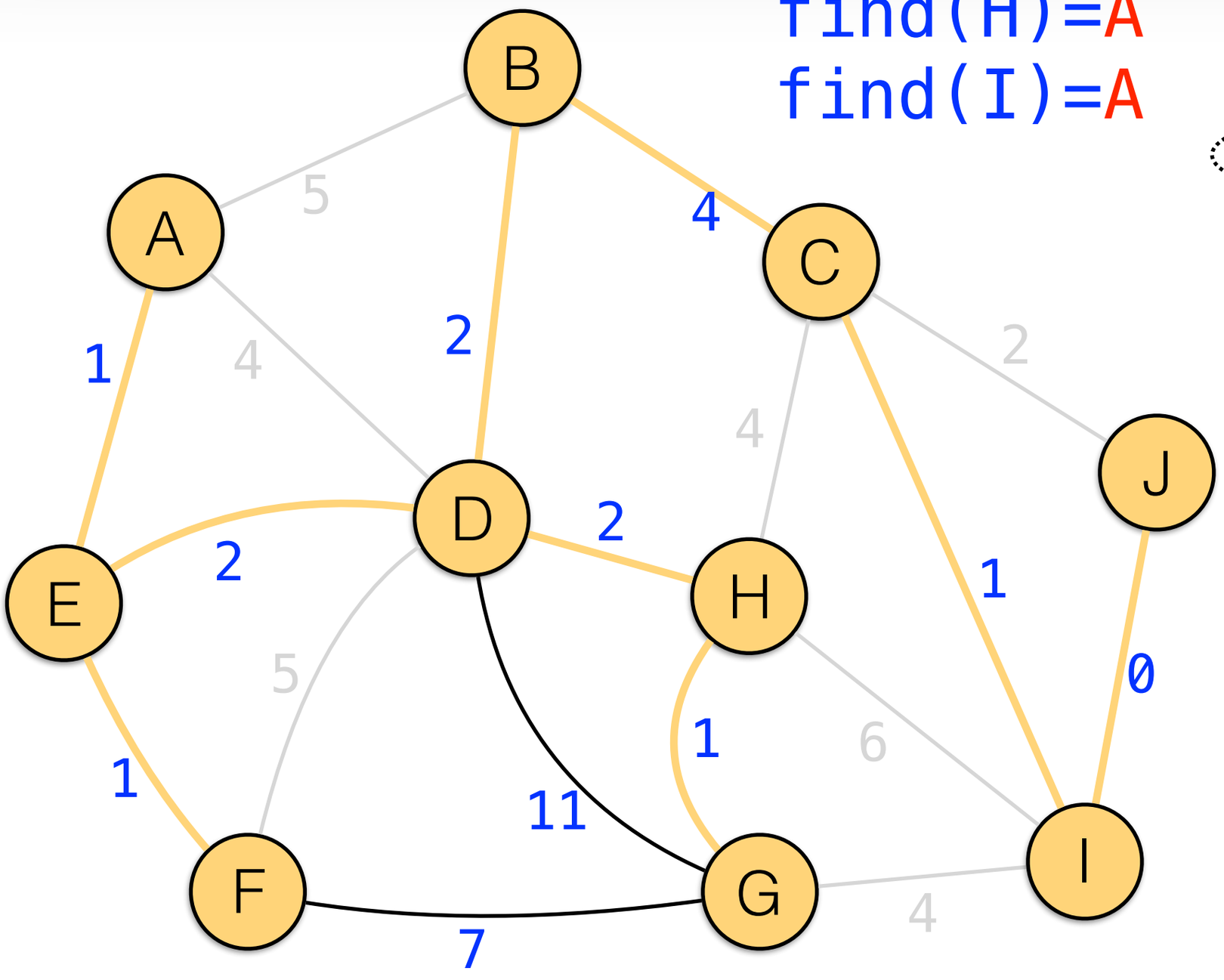
	I-J	A-E	C-I	E-F	G-H	B-D	C-J	D-E	D-H	A-D	B-C	C-H	G-I	A-B	D-F	H-I	F-G	D-G
d	0	1	1	1	1	2	2	2	2	4	4	4	4	5	5	6	7	11



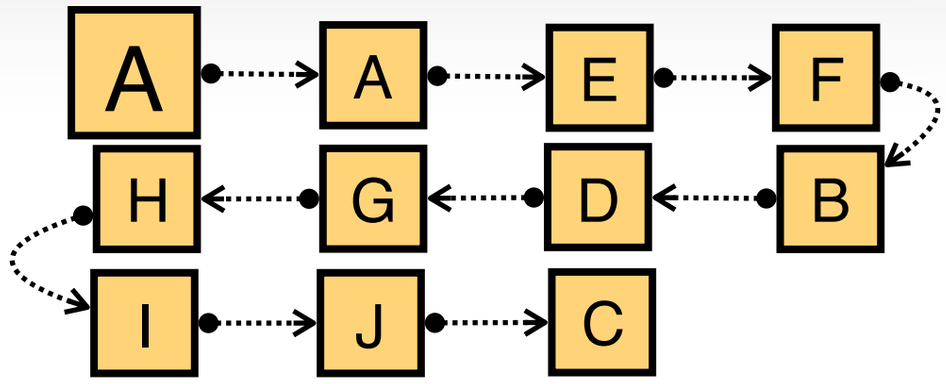


Kruskal MST

	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

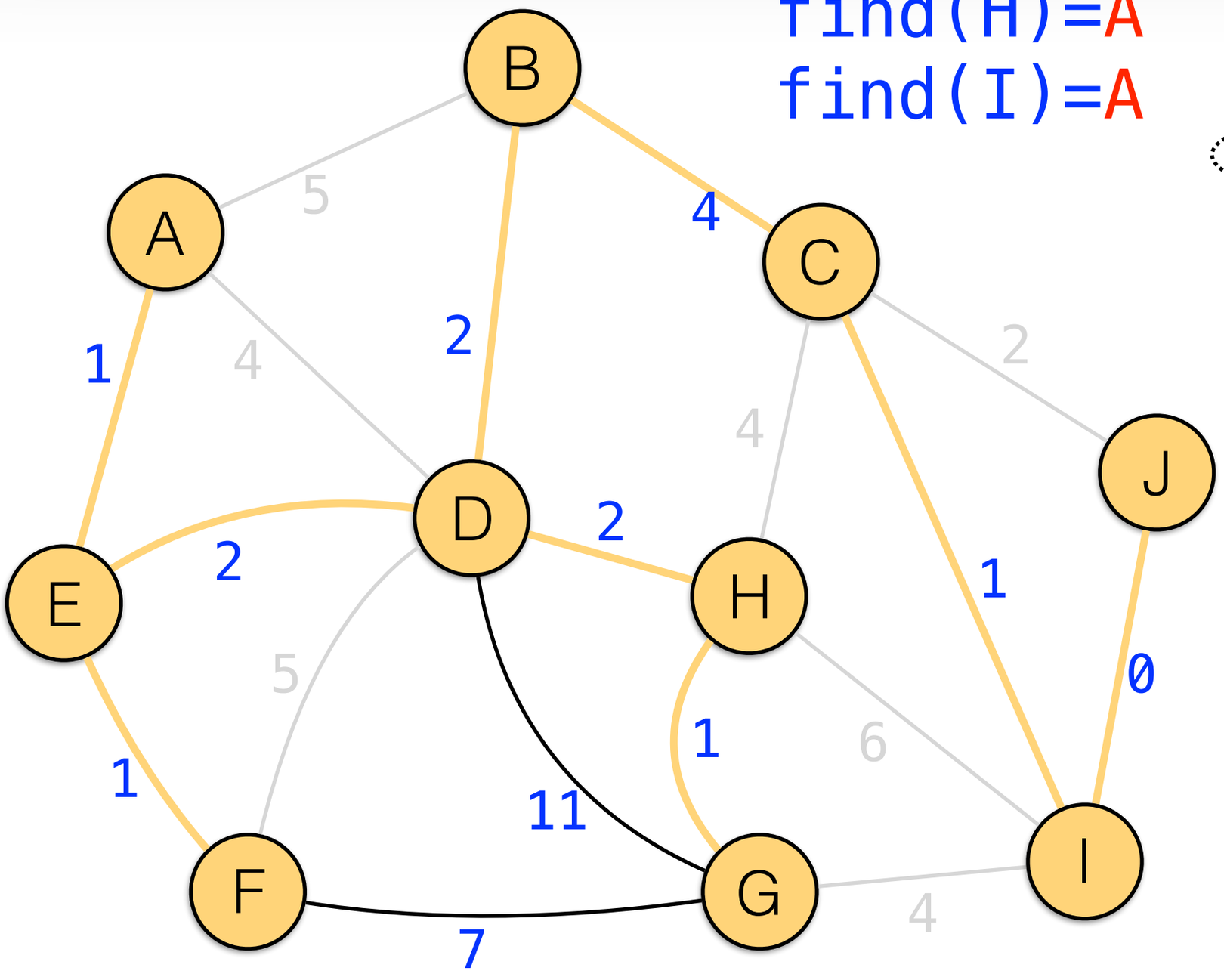



```
find(H)=A
find(I)=A
```

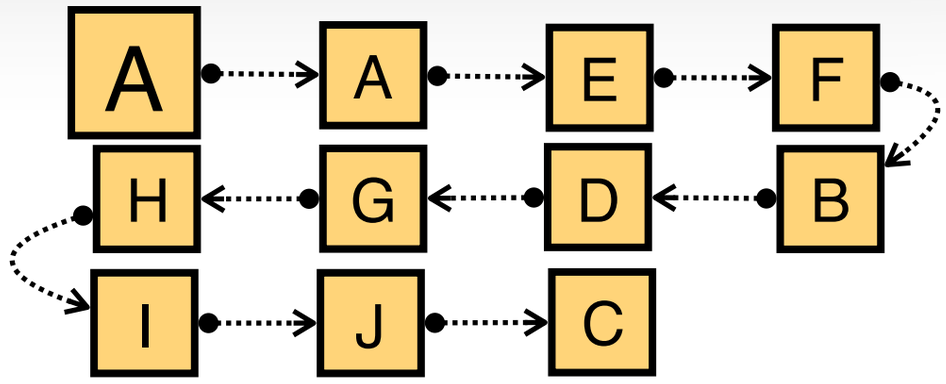


Kruskal MST

	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



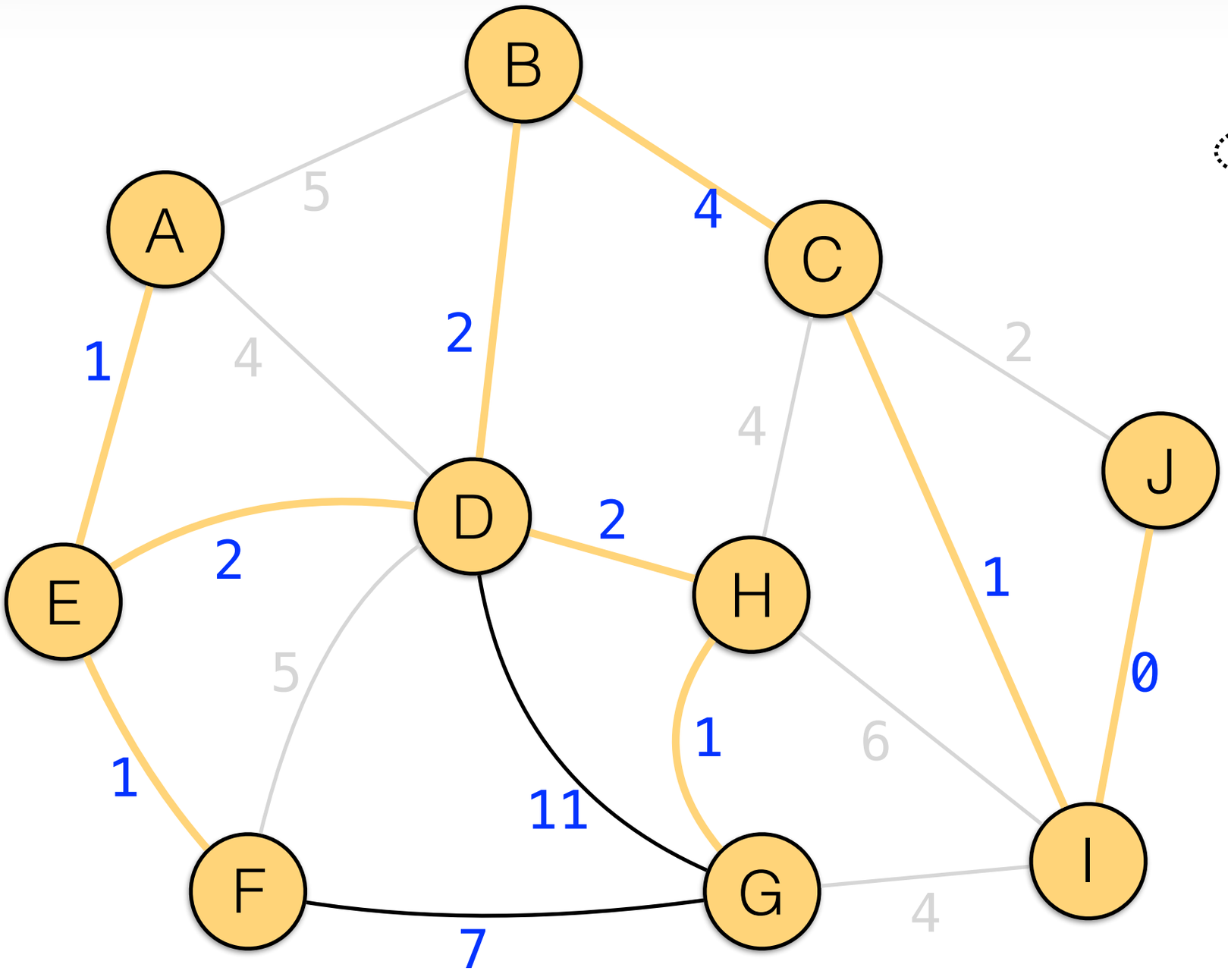

```
find(H)=A
find(I)=A
```

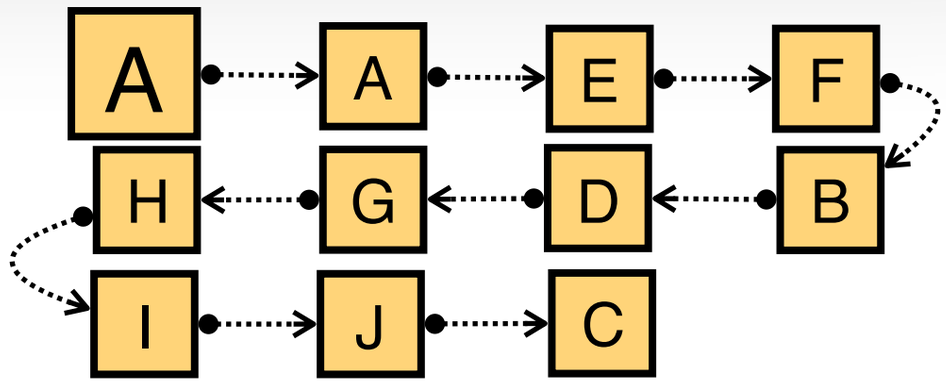


Do not add, it generates a cycle!

Kruskal MST

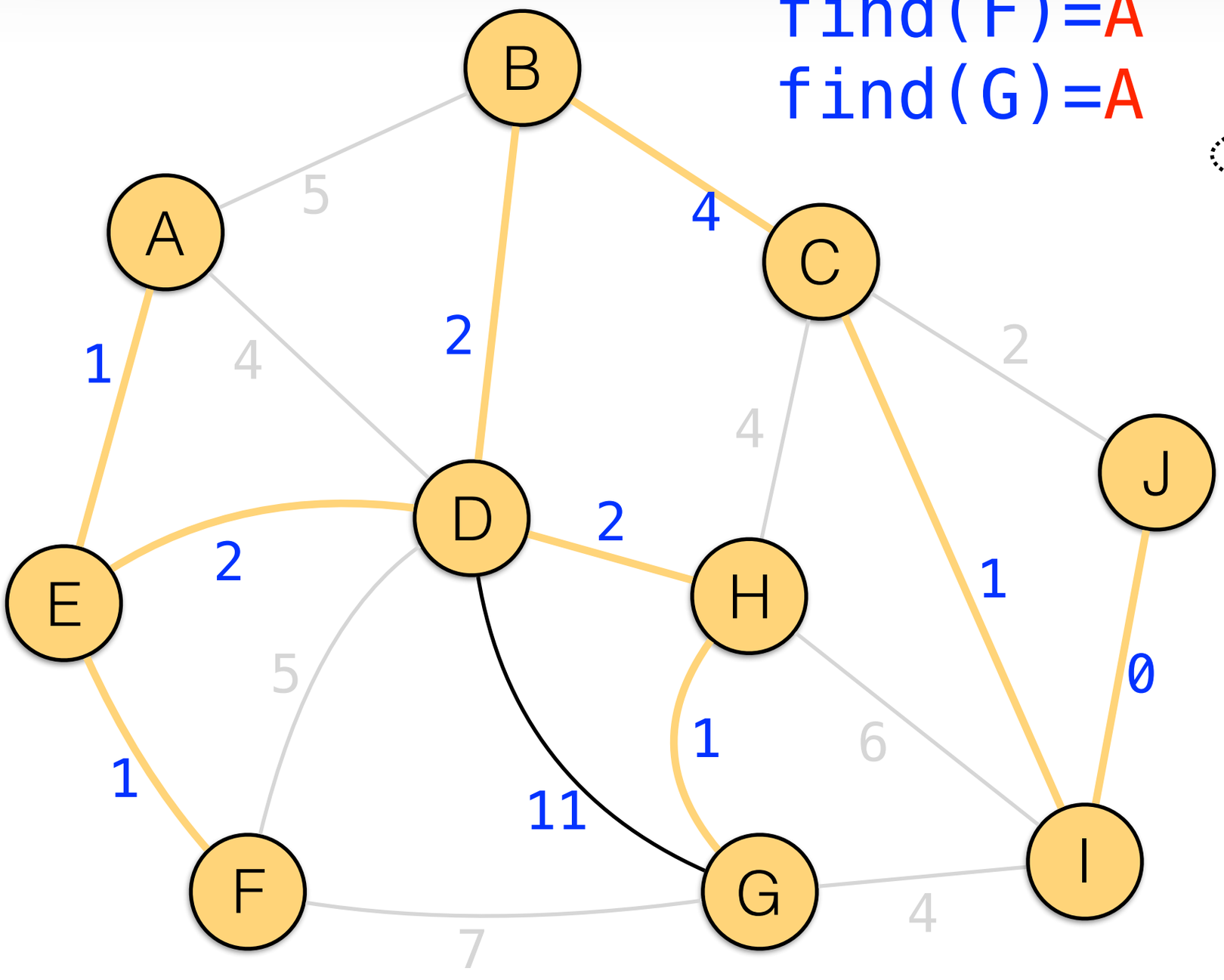
	I-J	A-E	C-I	E-F	G-H	B-D	C-J	D-E	D-H	A-D	B-C	C-H	G-I	A-B	D-F	H-I	F-G	D-G
d	0	1	1	1	1	2	2	2	2	4	4	4	4	5	5	6	7	11



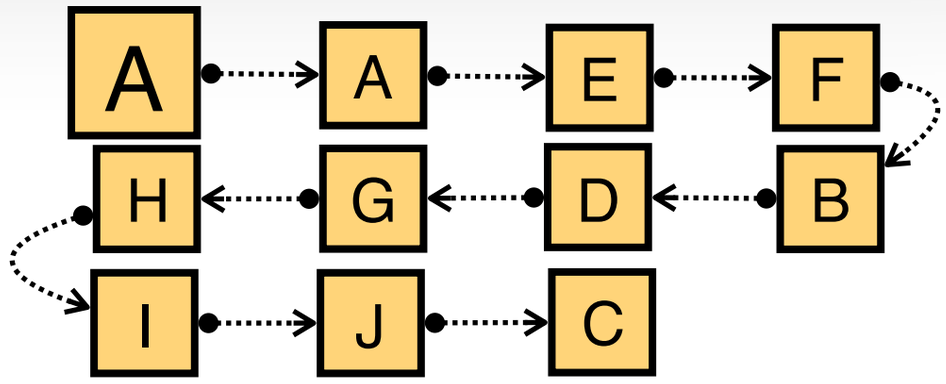


Kruskal MST

	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

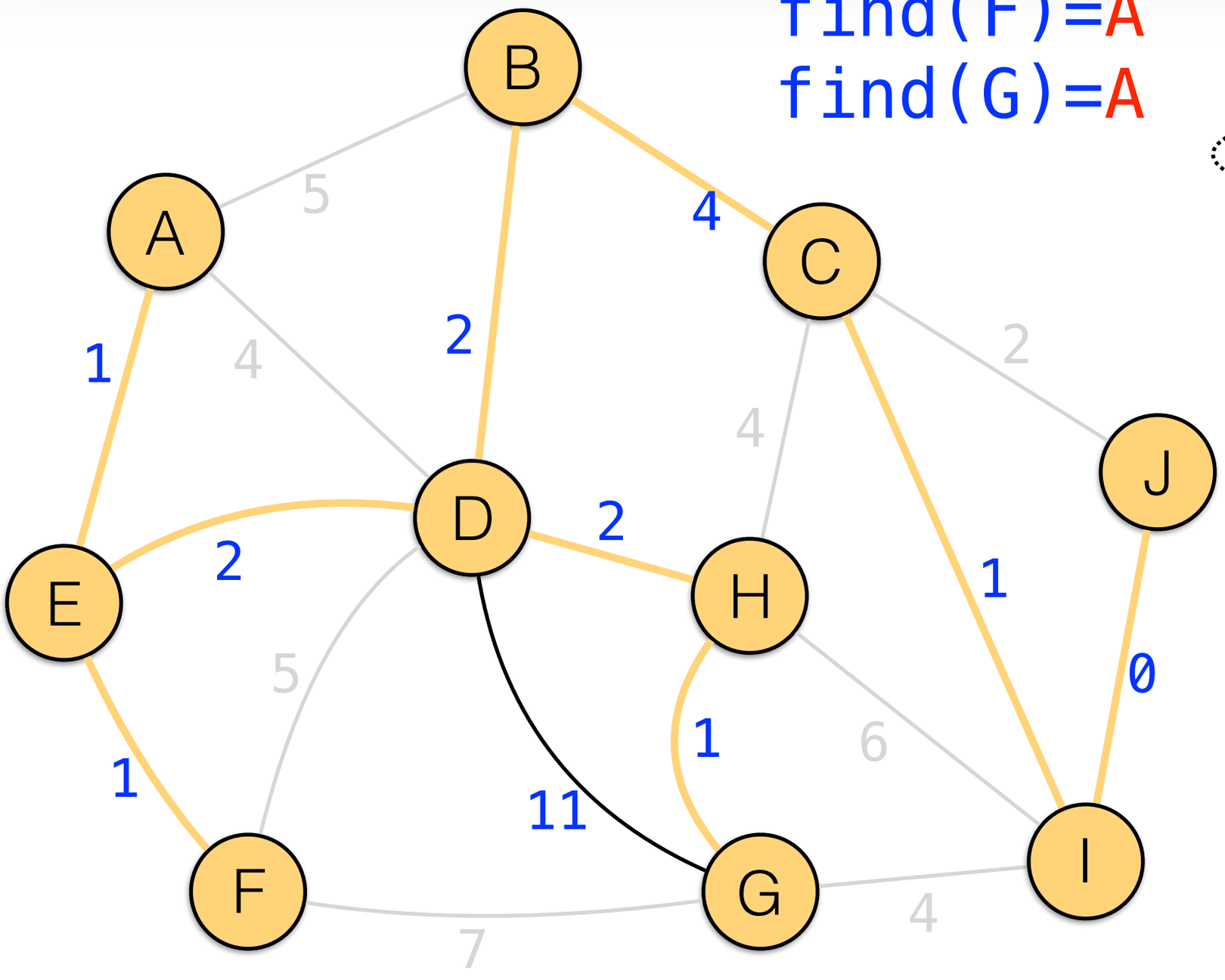



```
find(F)=A
find(G)=A
```

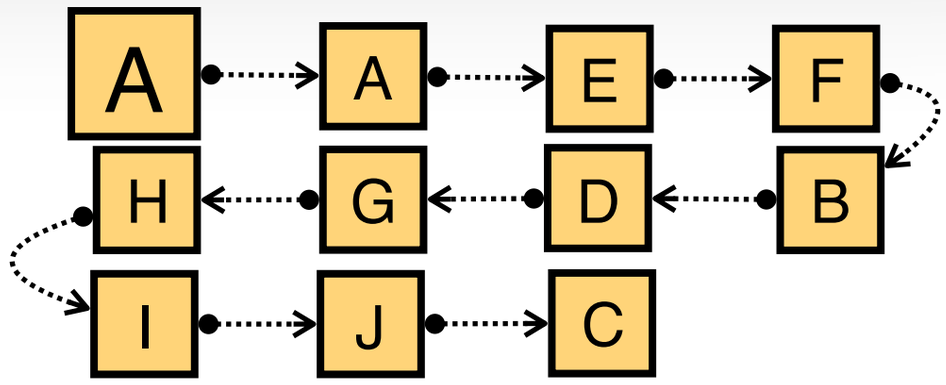


Kruskal MST

	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



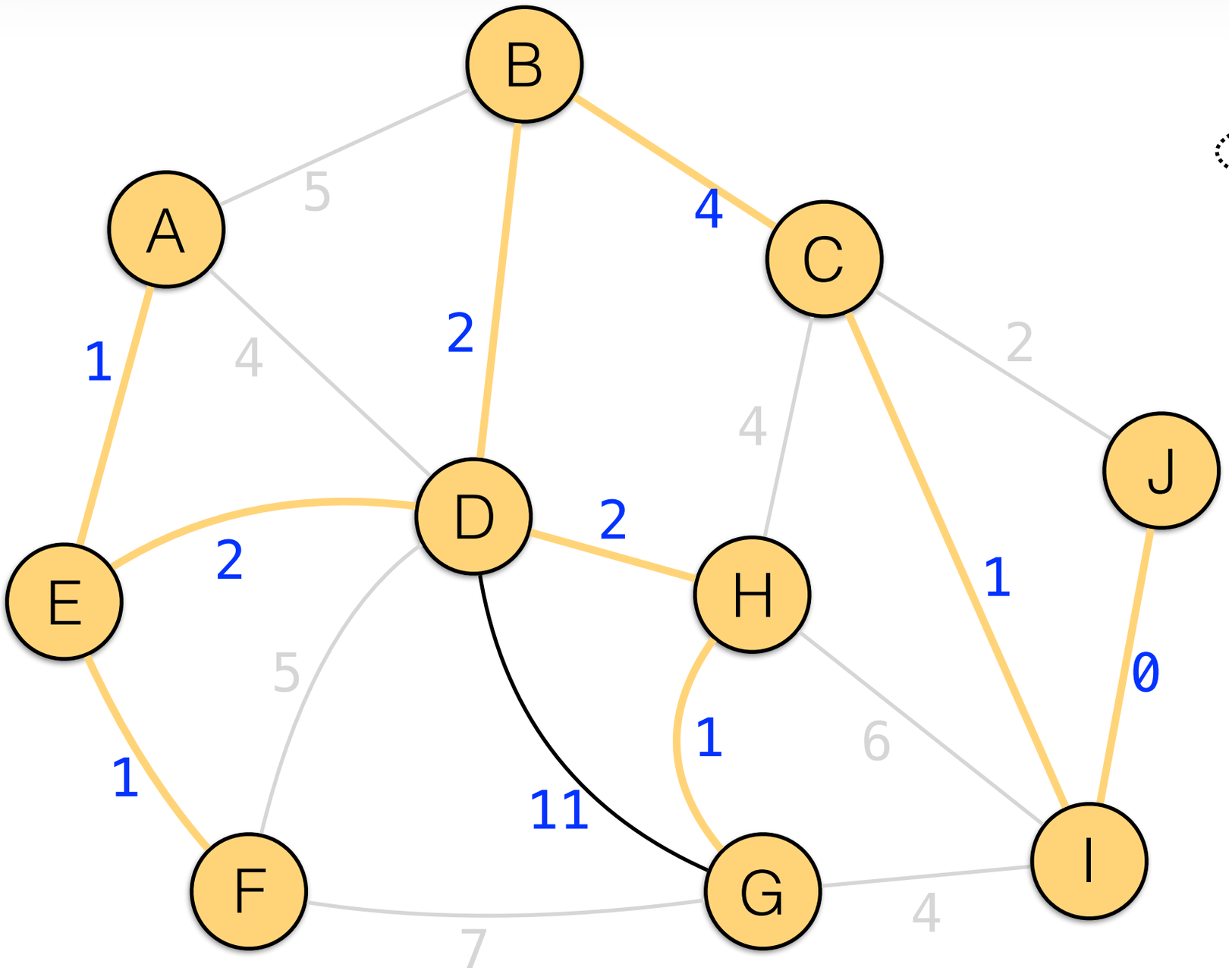

```
find(F)=A
find(G)=A
```

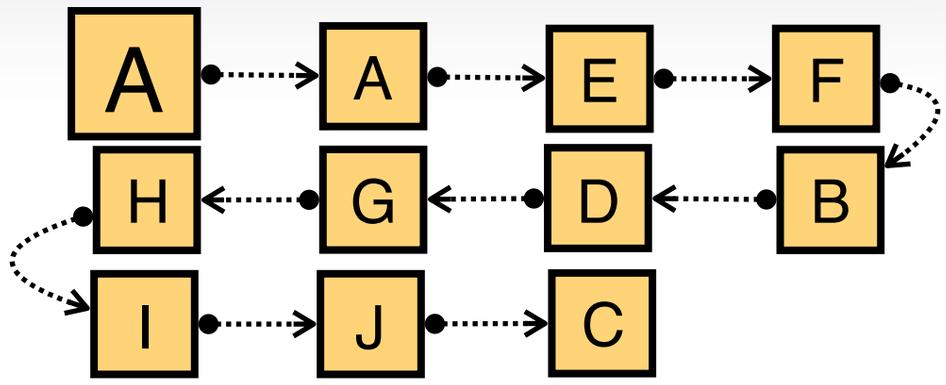


Do not add, it generates a cycle!

Kruskal MST

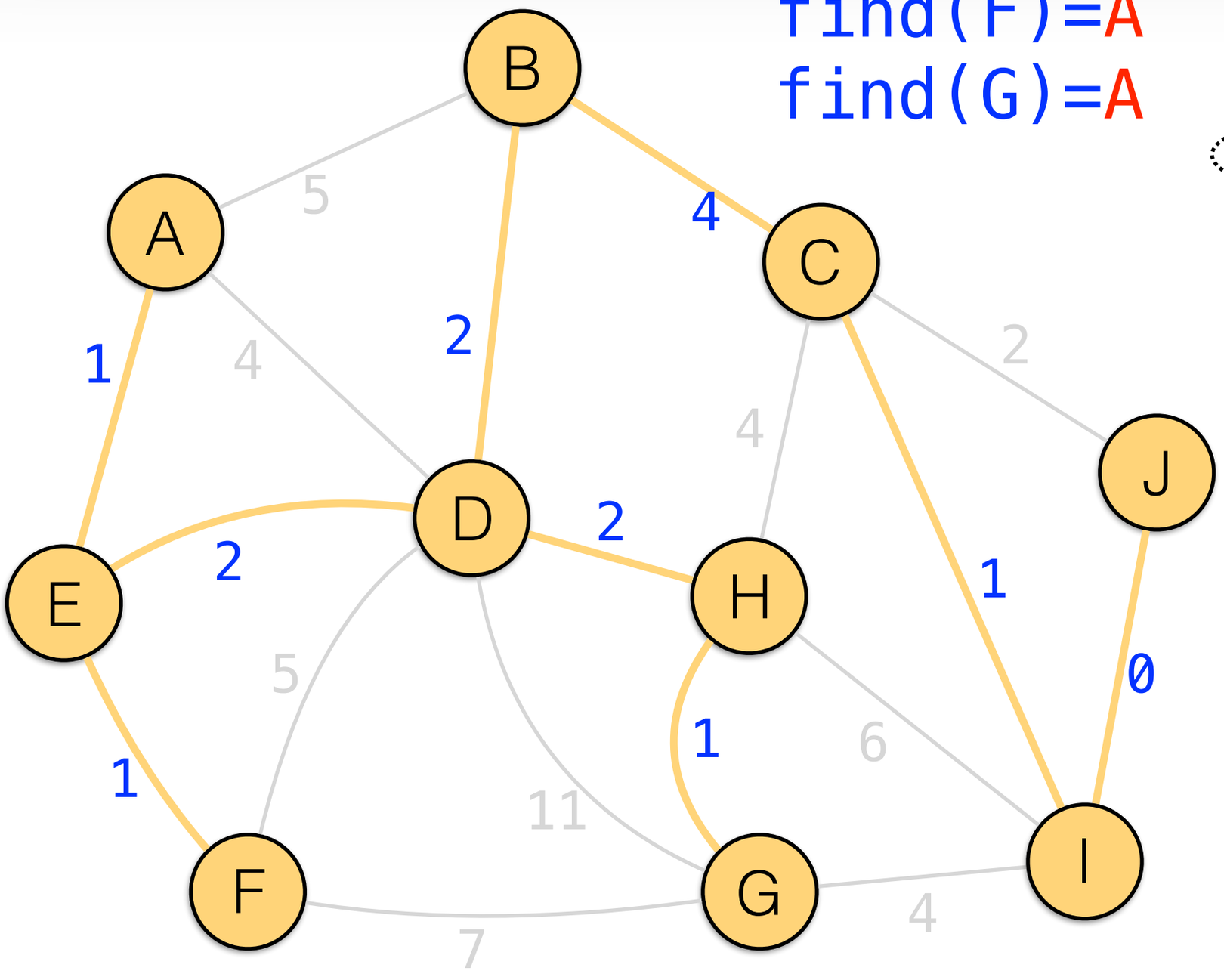
	I-J	A-E	C-I	E-F	G-H	B-D	C-J	D-E	D-H	A-D	B-C	C-H	G-I	A-B	D-F	H-I	F-G	D-G
d	0	1	1	1	1	2	2	2	2	4	4	4	4	5	5	6	7	11



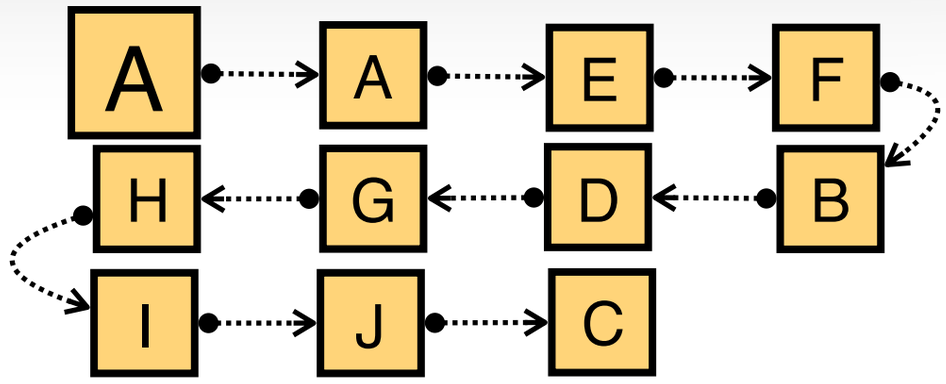


Kruskal MST

	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

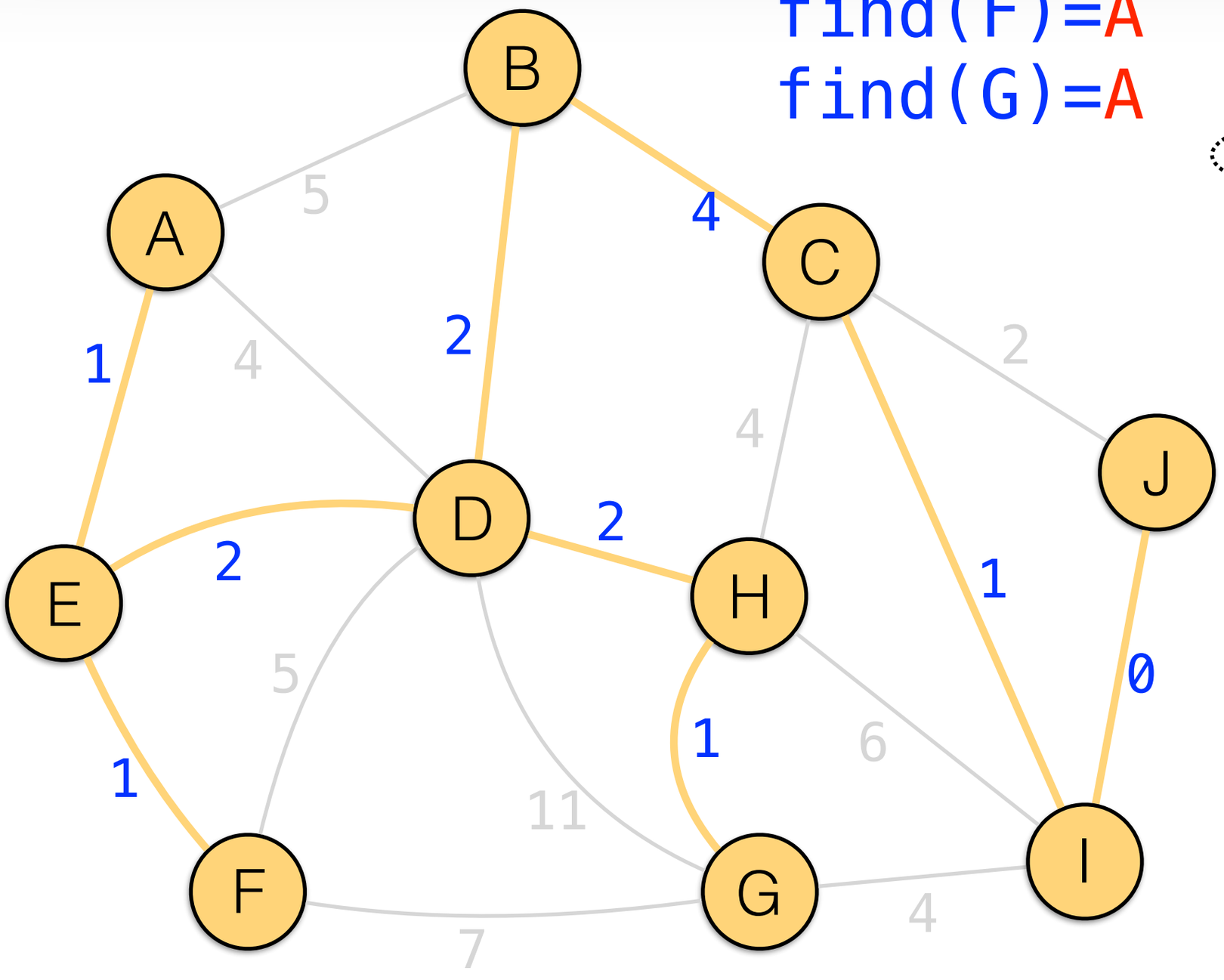



```
find(F)=A
find(G)=A
```

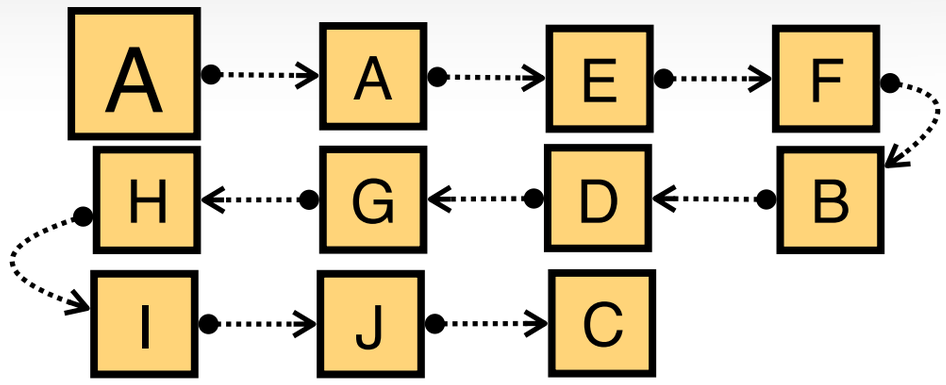


Kruskal MST

	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(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.

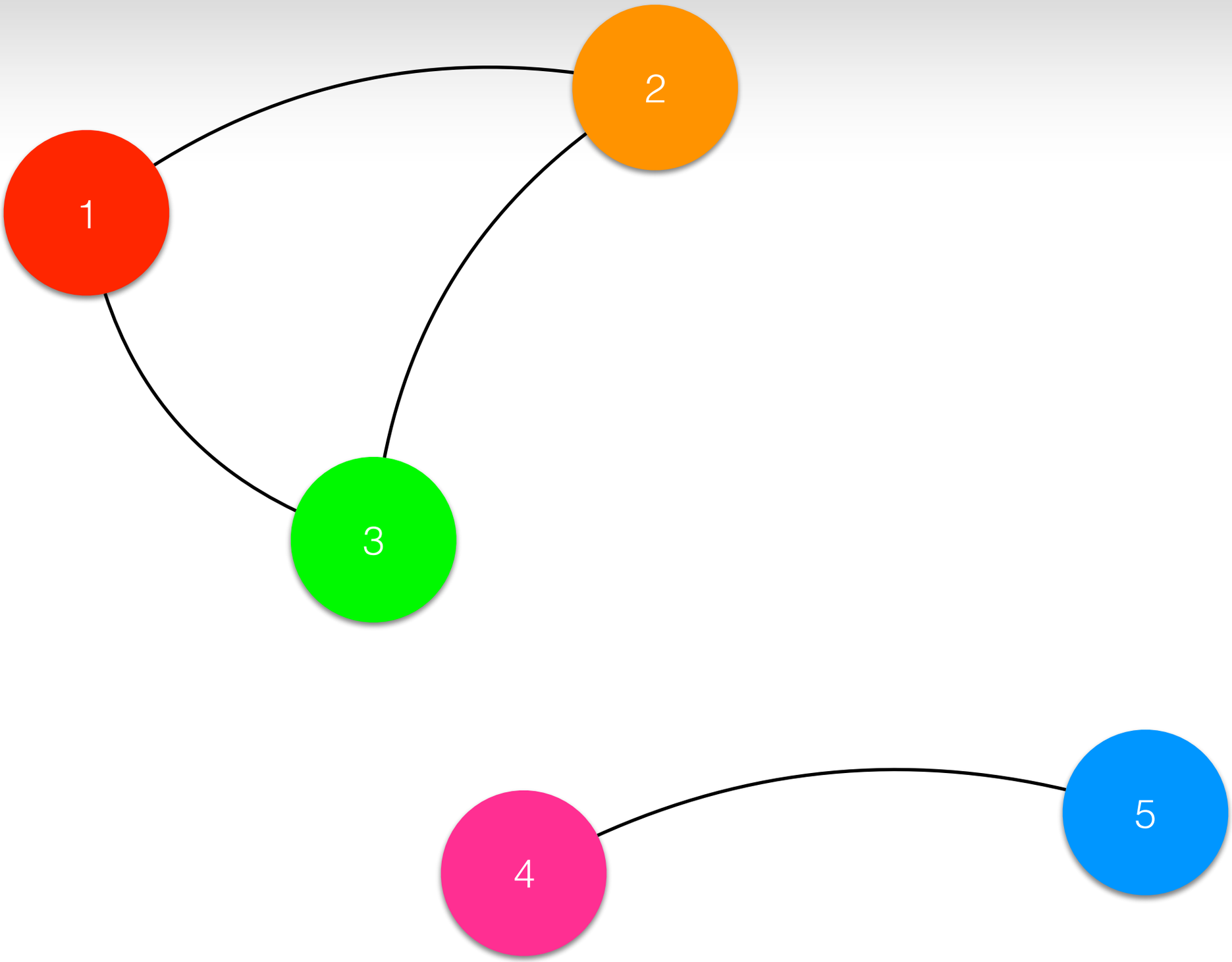
Output

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

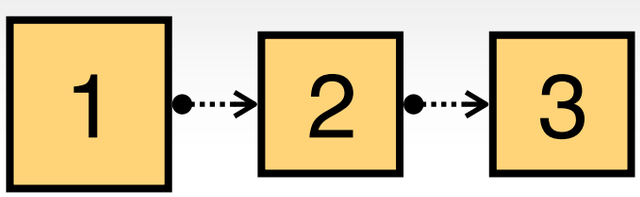
Constraints

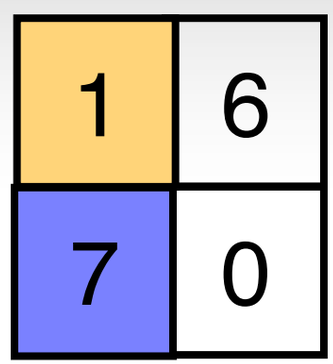
- $1 \leq Xj, Yj \leq N$
- Roads' numbers are 1-indexed.
- There is no road that gets removed twice or more.
- $1 \leq Pi \leq 105$
- Subtask 1 (30 points) : $1 \leq N, M, Q \leq 103$
- Subtask 2 (70 points) : $1 \leq N, M, Q \leq 5 \times 105$

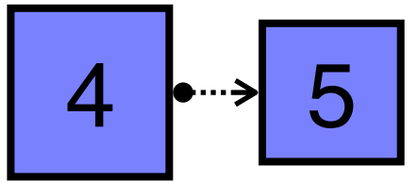
https://www.codechef.com/problems/ABROADS

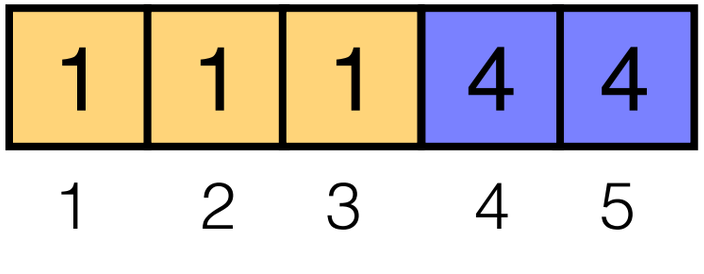


https://www.codechef.com/problems/ABROADS









https://codeforces.com/problemset/problem/217/A

Bajtek is learning to skate on ice. He's a beginner, so his only mode of transportation is pushing off from a snow drift to the north, east, south or west and sliding until he lands in another snow drift. He has noticed that in this way it's impossible to get from some snow drifts to some other by any sequence of moves. He now wants to heap up some additional snow drifts, so that he can get from any snow drift to any other one. He asked you to find the minimal number of snow drifts that need to be created.

We assume that Bajtek can only heap up snow drifts at integer coordinates.

https://codeforces.com/problemset/problem/217/A

Input

The first line of input contains a single integer n ($1 \le n \le 100$) — the number of snow drifts. Each of the following n lines contains two integers xi and yi ($1 \le xi, yi \le 1000$) — the coordinates of the i-th snow drift.

Note that the north direction coincides with the direction of Oy axis, so the east direction coincides with the direction of the Ox axis. All snow drift's locations are distinct.

Output

Output the minimal number of snow drifts that need to be created in order for Bajtek to be able to reach any snow drift from any other one.

https://codeforces.com/problemset/problem/217/A