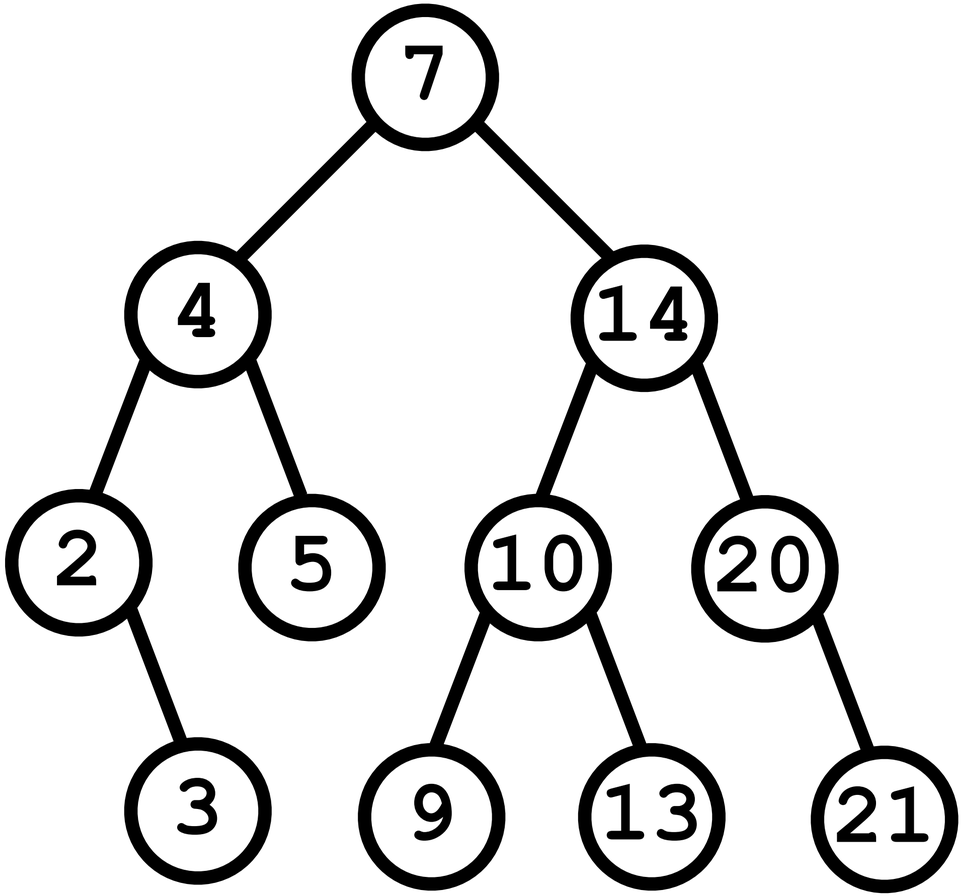
COSC 2436: Final Exam Review

J. Eoin Donovan - 2023

BST #1

Write the traversal name under each different type of traversal.



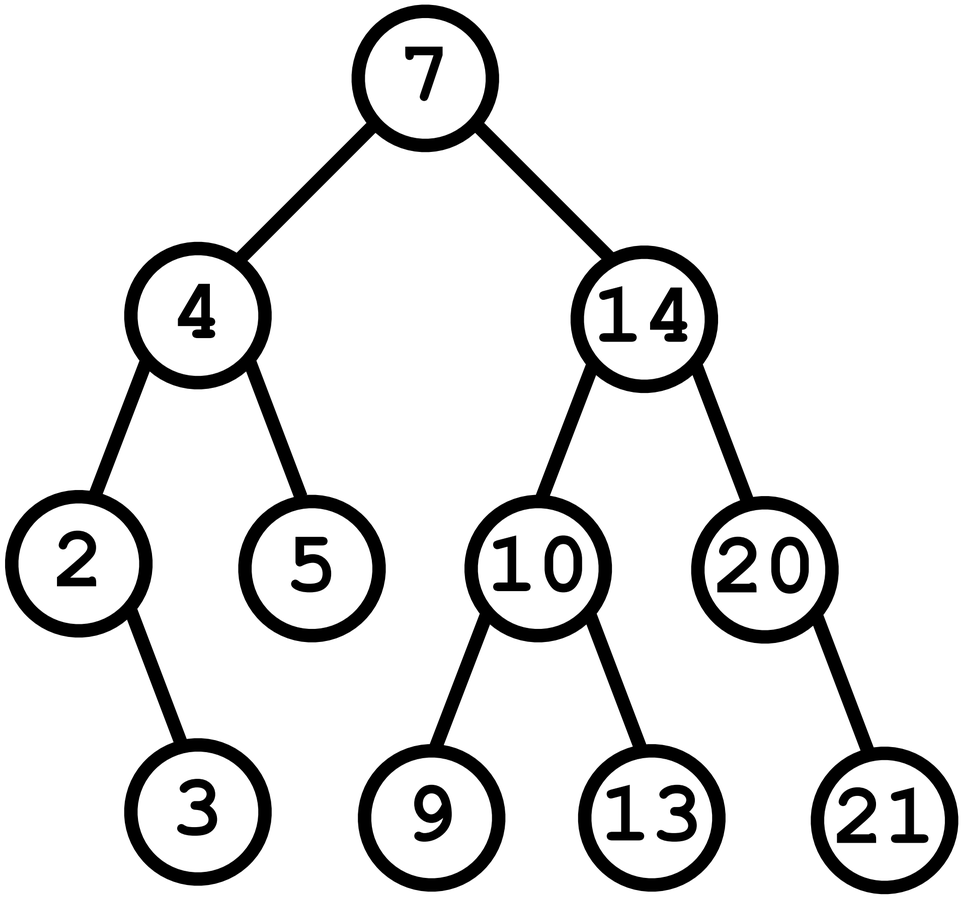
3, 2, 5, 4, 9, 13, 10, 21, 20, 14, 7

2, 3, 4, 5, 7, 9, 10, 13, 14, 20, 21

7, 4, 2, 3, 5, 14, 10, 9, 13, 20, 21

BST #1

Write the traversal name under each different type of traversal.



3, 2, 5, 4, 9, 13, 10, 21, 20, 14, 7

Postorder

2, 3, 4, 5, 7, 9, 10, 13, 14, 20, 21

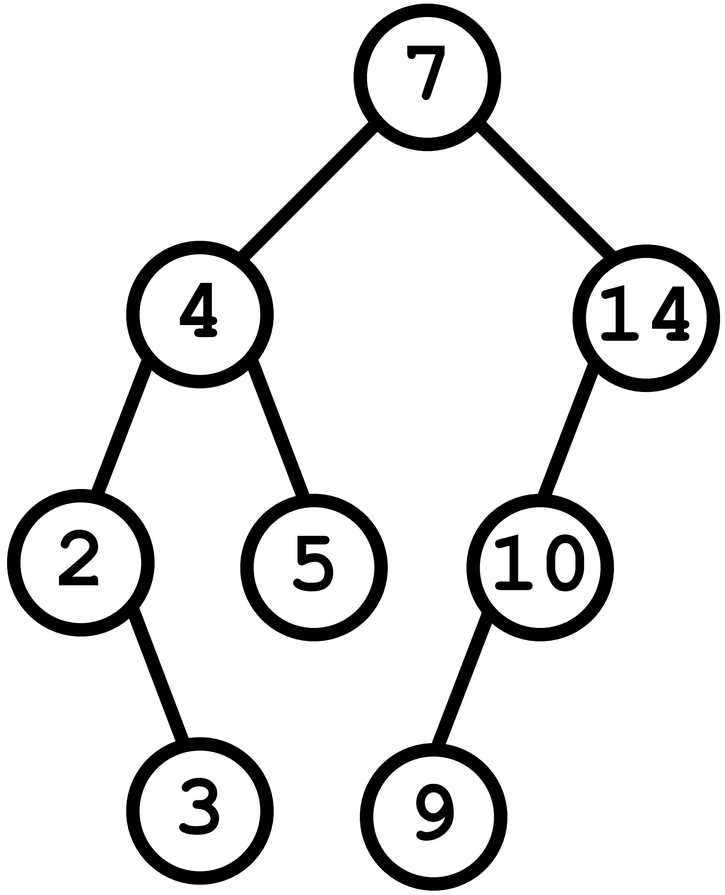
Inorder

7, 4, 2, 3, 5, 14, 10, 9, 13, 20, 21

Preorder

BST #2

Answer the following questions about the BST below.



What is the height of the BST?

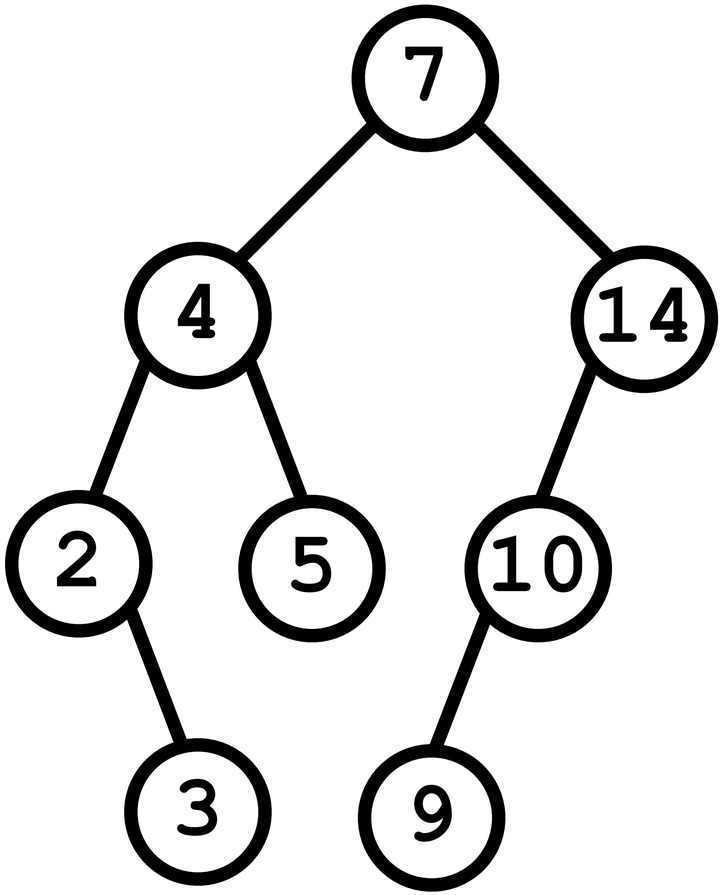
What is the depth of node 10?

What is the lowest common ancestor of nodes 3 and 5?

What nodes make up level 3 of the BST?

BST #2

Answer the following questions about the BST below.



What is the height of the BST?

3

What is the depth of node 10?

2

What is the lowest common ancestor of nodes 3 and 5?

node 4

What nodes make up level 3 of the BST?

node 2, node 5, and node 10

BST #3

Complete the function `int getSize(node *root)` which returns the size (number of nodes) in a BST.

```
struct node{
   int value;
   node *left;
   node *right;
};

int getSize(node *root){



}
```

BST #3

```
int getSize(node *root){
  if(root == nullptr){
    return 0;
  }
  else if(root->left == nullptr && root->right == nullptr){
    return 1;
  }
  else{
    return 1 + getSize(root->left) + getSize(root->right);
  }
}
```

BST #4

Complete the function `node *getNode(node *root, int value)` which returns the node containing `value`. Your function must use recursion (no iteration).

```
struct node{
  int value;
  node *left;
  node *right;
};

node *getNode(node *root, int value){



}
```

BST #4

```
node *getNode(node *root, int value){
  if(root == nullptr){
    return nullptr;
  }
  else if(value < root->value){
    return getNode(root->left, value);
  }
  else if(value > root->value){
    return getNode(root->right, value);
  }
  else{
    return root;
  }
}
```

BST #5

Complete the function `node *getNode(node *root, int value)` which returns the node containing `value`. Your function must use iteration (no recursion).

```
struct node{
  int value;
  node *left;
  node *right;
};

node *getNode(node *root, int value){



}
```

BST #5

```
node *getNode(node *root, int value){
  while(root != nullptr){
    if(root->value == value){
      return root;
    }
    else if(root->value < value){
      root = root->left;
    }
    else if(root->value > value){
      root = root->right;
    }
  }
  return root;
}
```

BST #6

Complete the function `int getDepth(node *root, int value, int depth)` which finds the depth of the node containing `value`. Assume depth is passed in as 0. Return -1 if `value` does not exists.

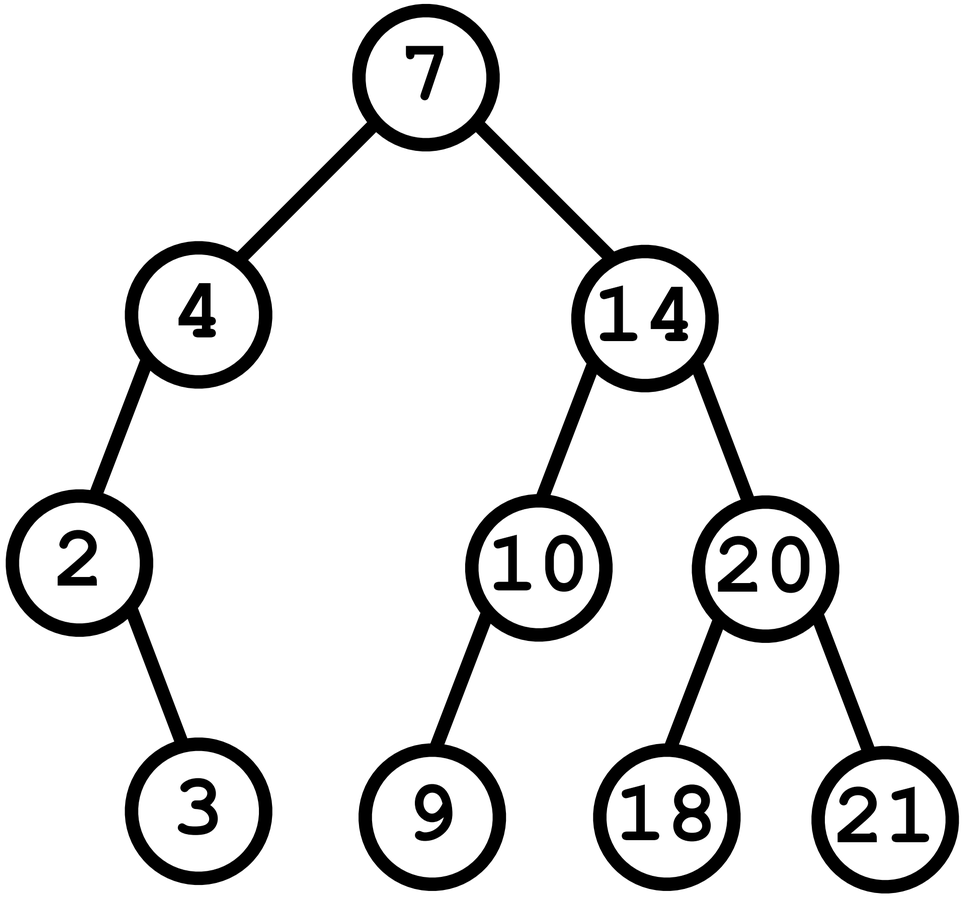
```
struct node{
   int value;
   node *left;
   node *right;
};

int getDepth(node *root, int value, int depth){

}
```

BST #6

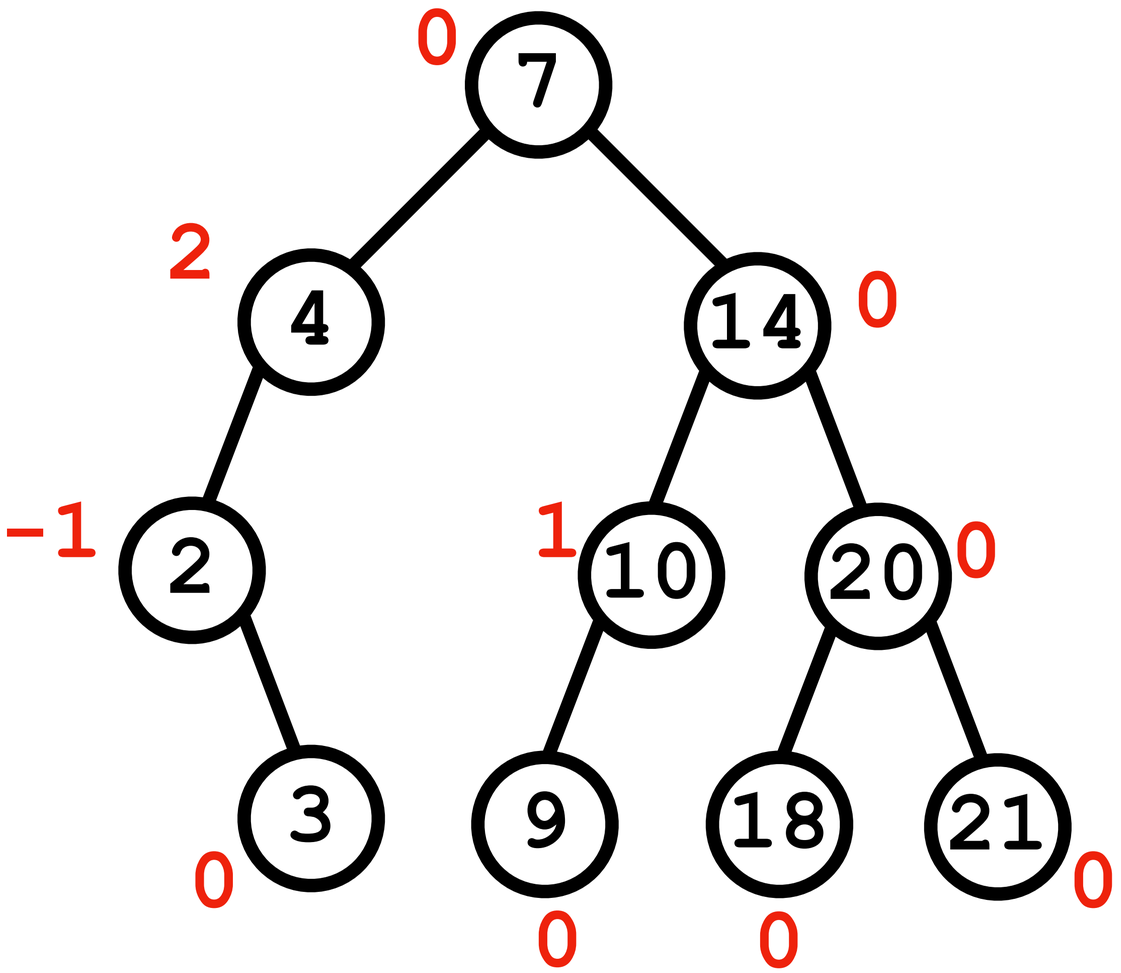
```
int getDepth(node *root, int value, int depth){
  if(root == nullptr){
    return -1;
  }
  else if(root->value == value){
    return depth;
  }
  else if(value < root->value){
    return getDepth(root->left, value, depth + 1);
  }
  else{
    return getDepth(root->right, value, depth + 1);
  }
}
```

AVL Tree #1

Give the balance factor of every node in the following BST and indicate which nodes are unbalanced.



AVL Tree #1



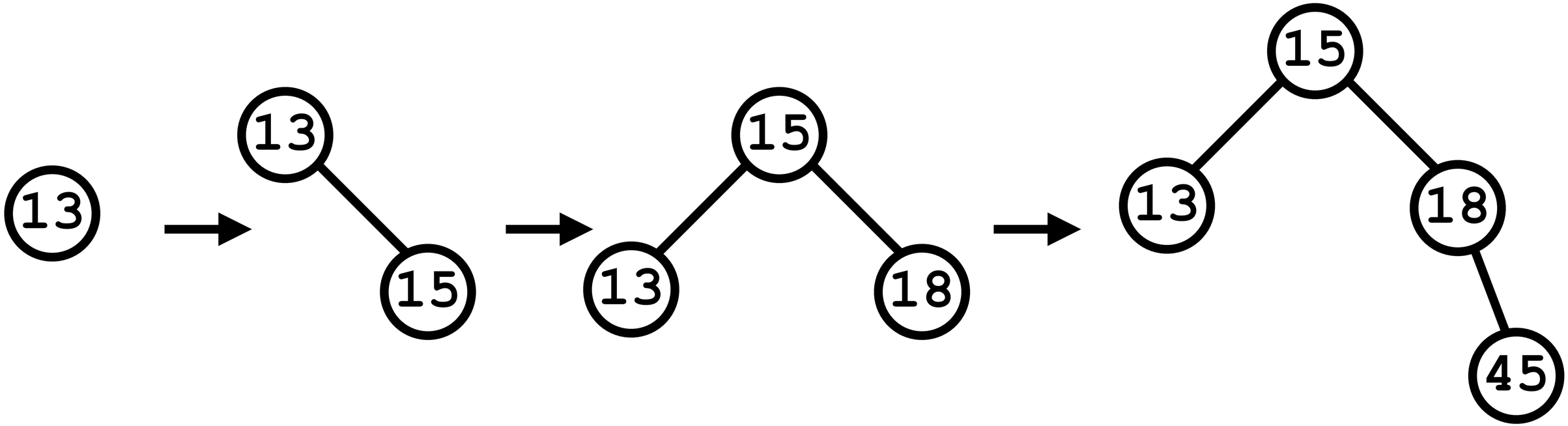
node 4 is unbalanced

AVL Tree #2

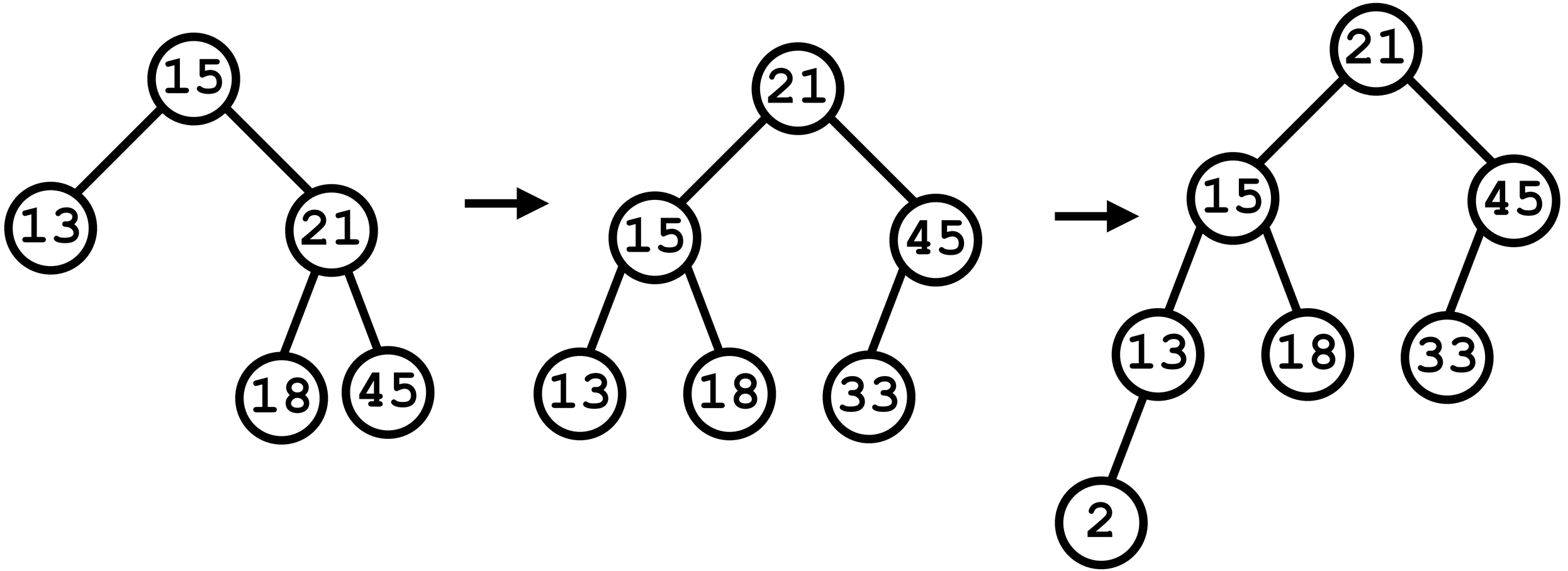
Insert the following values into an AVL Tree in the order in which they appear.

13, 15, 18, 45, 21, 33, 2, 1

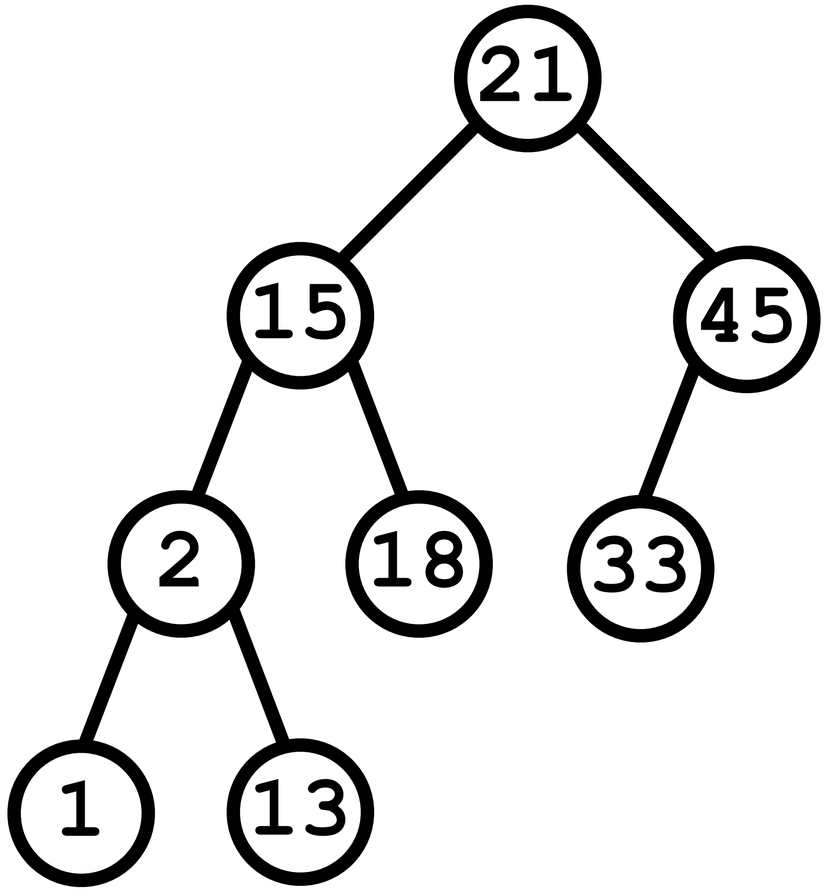
AVL Tree #2



AVL Tree #2

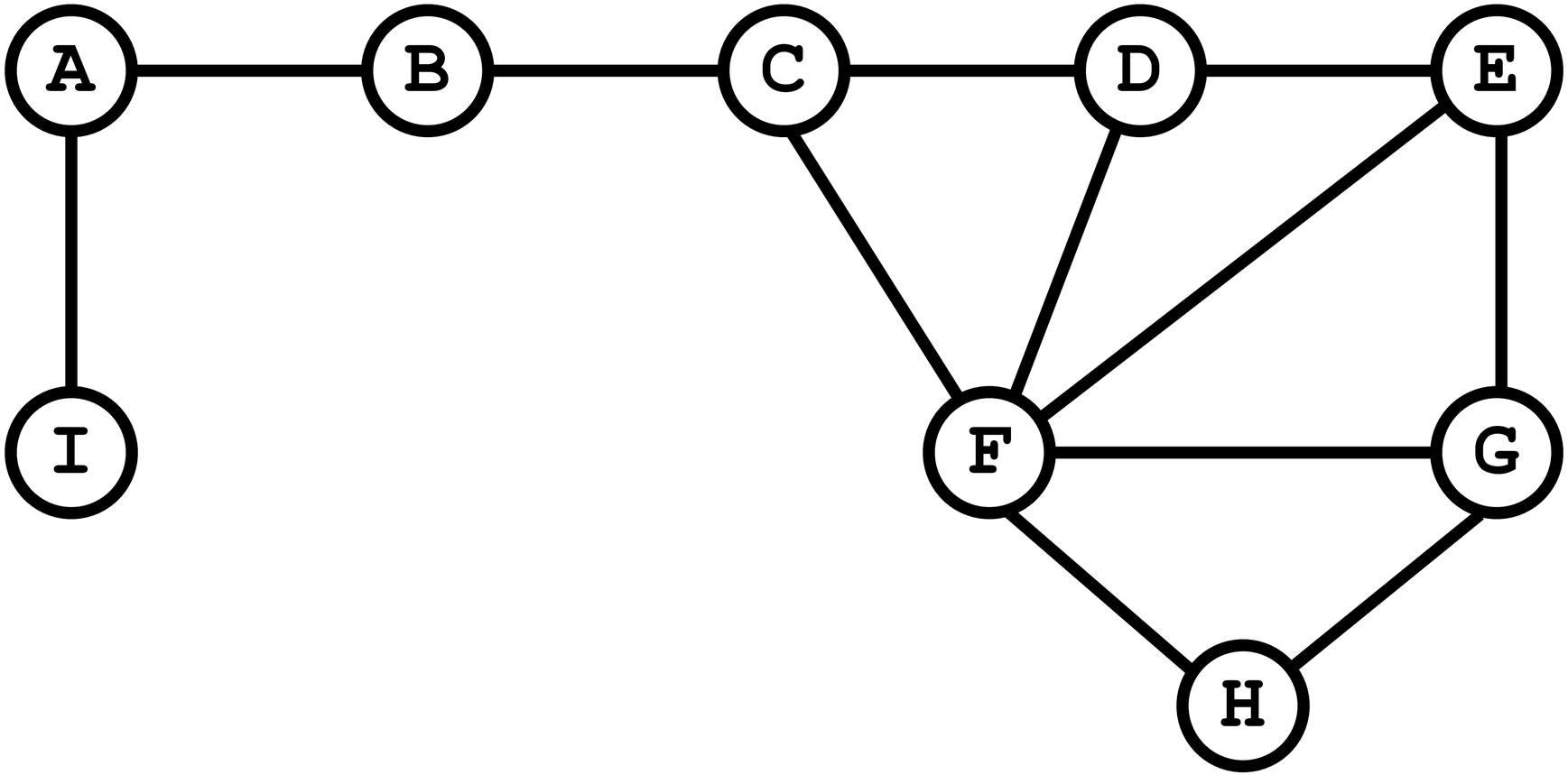


AVL Tree #2

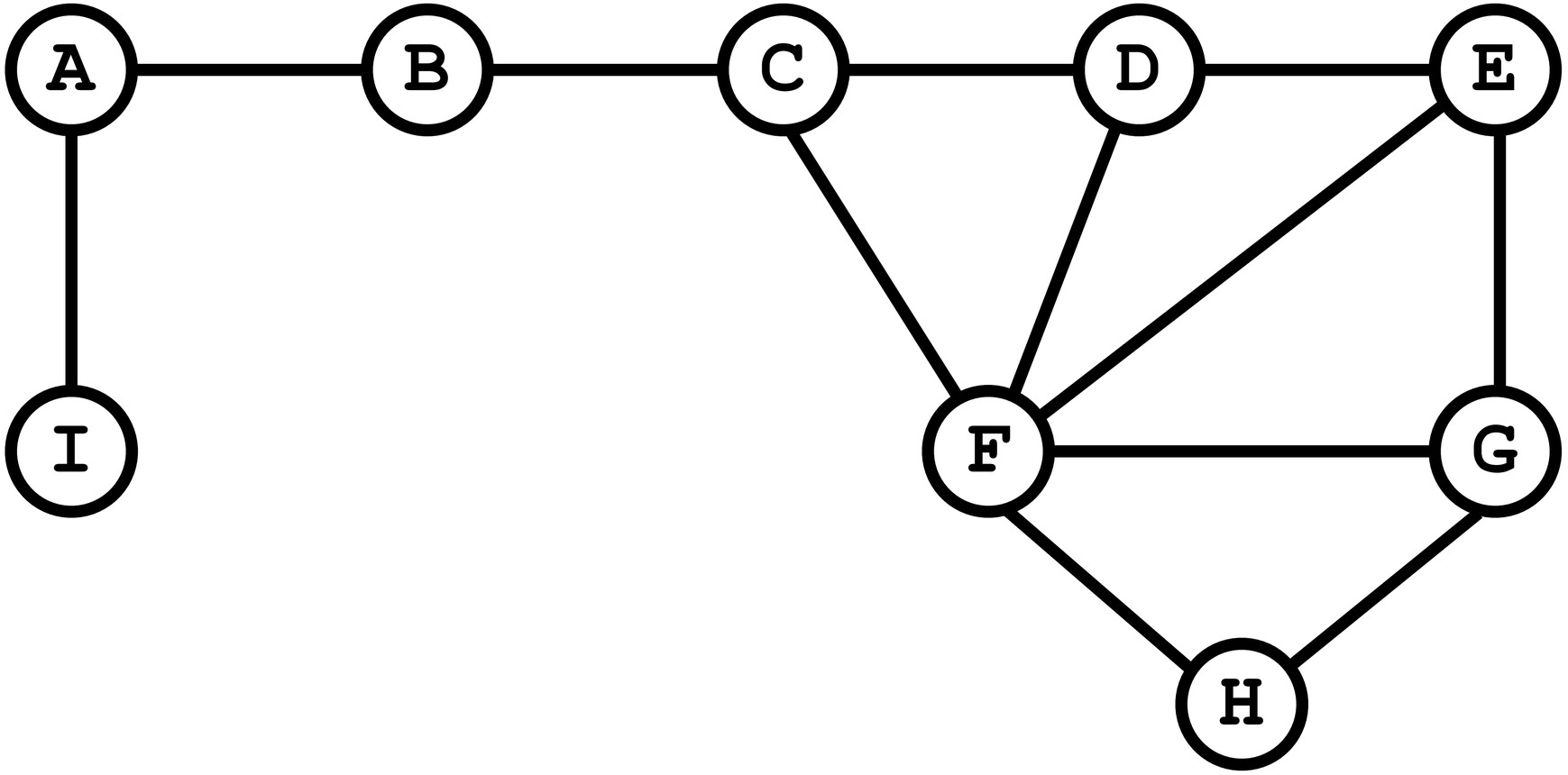


Graphs #1

Perform Depth First Traversal and Breadth First Traversal starting from vertex A on the graph below.



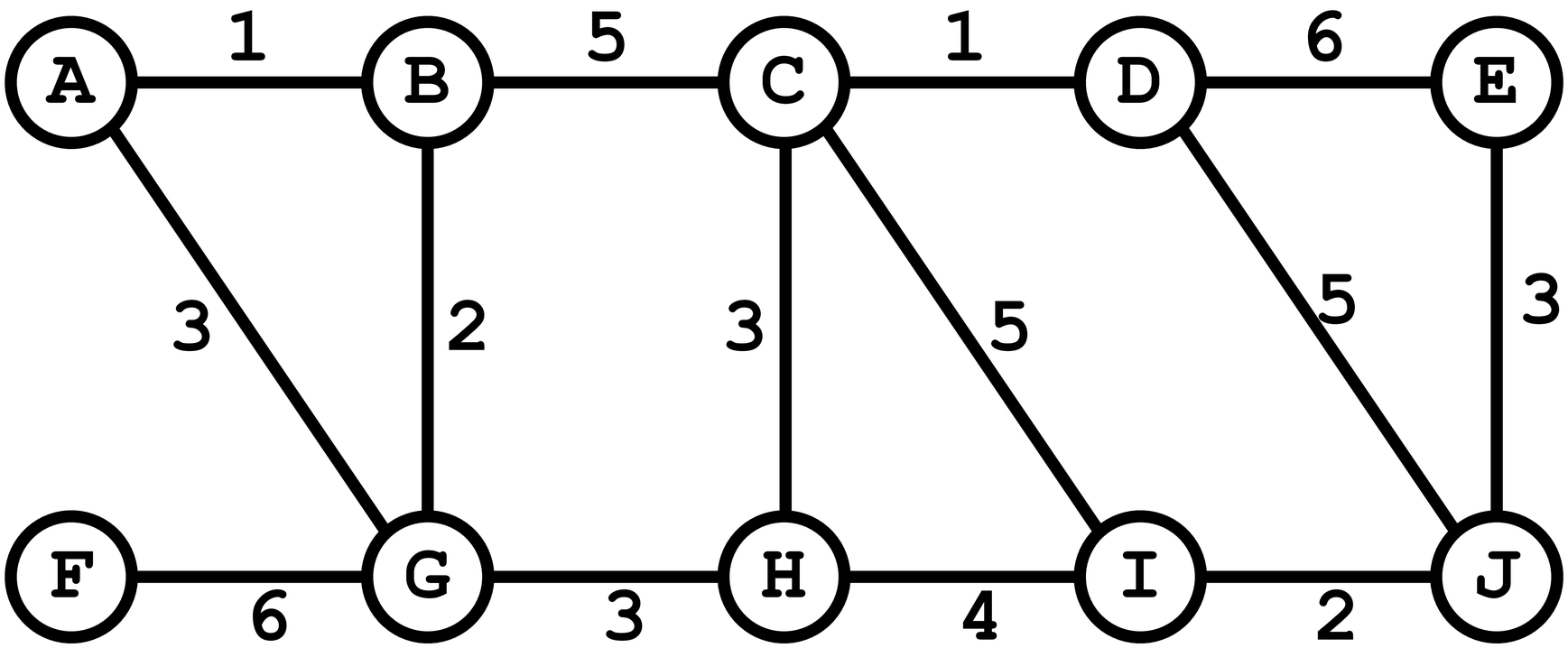
Graphs #1



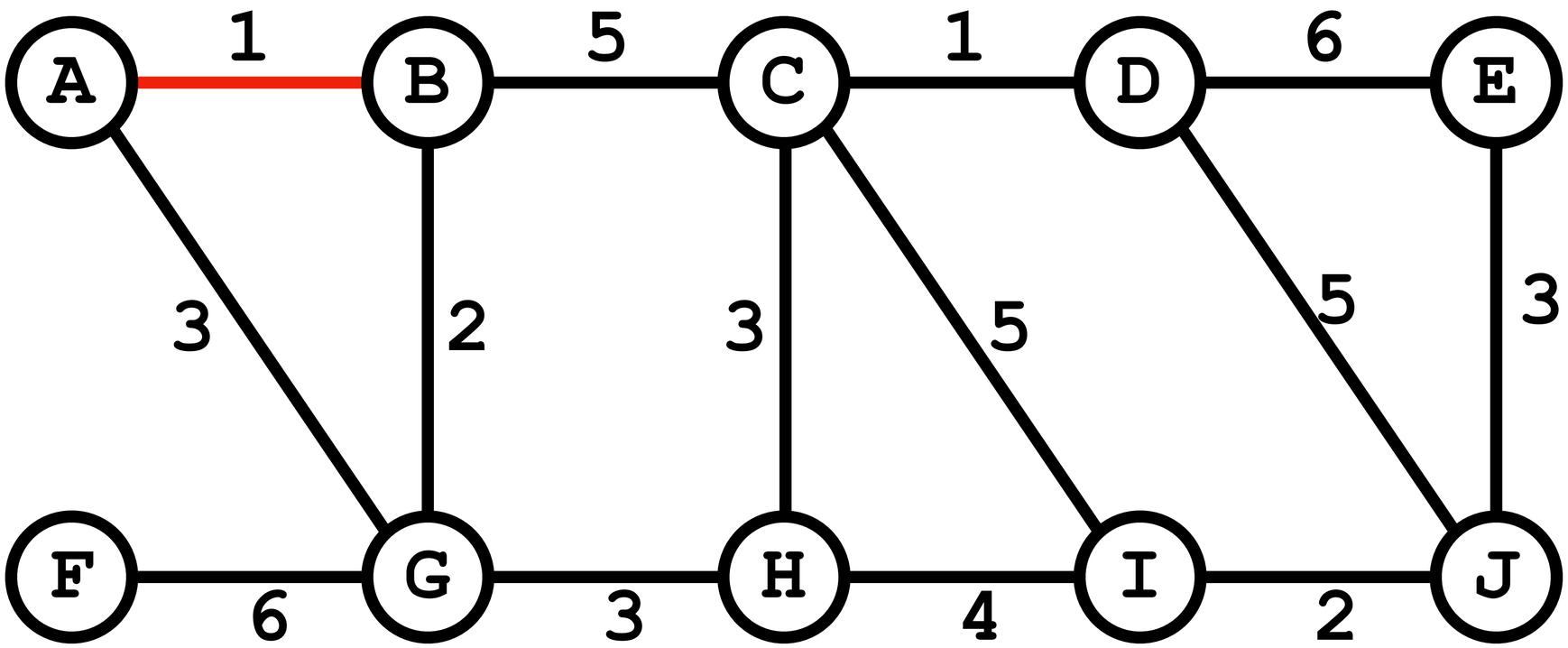
Depth First Traversal: A, B, C, D, E, F, G, H, I
Breadth First Traversal: A, B, I, C, D, F, E, G, H

Graphs #2

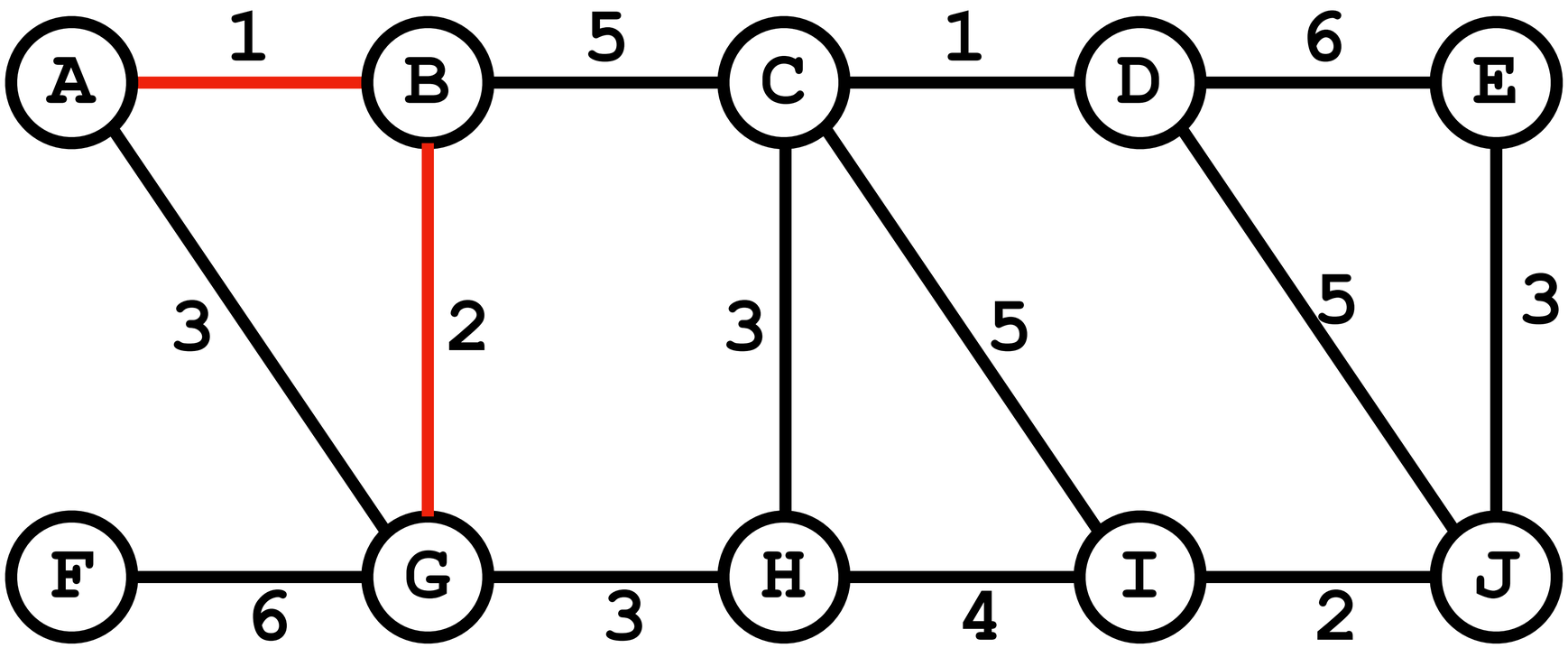
Draw the MST using Prim's algorithm starting from vertex A.



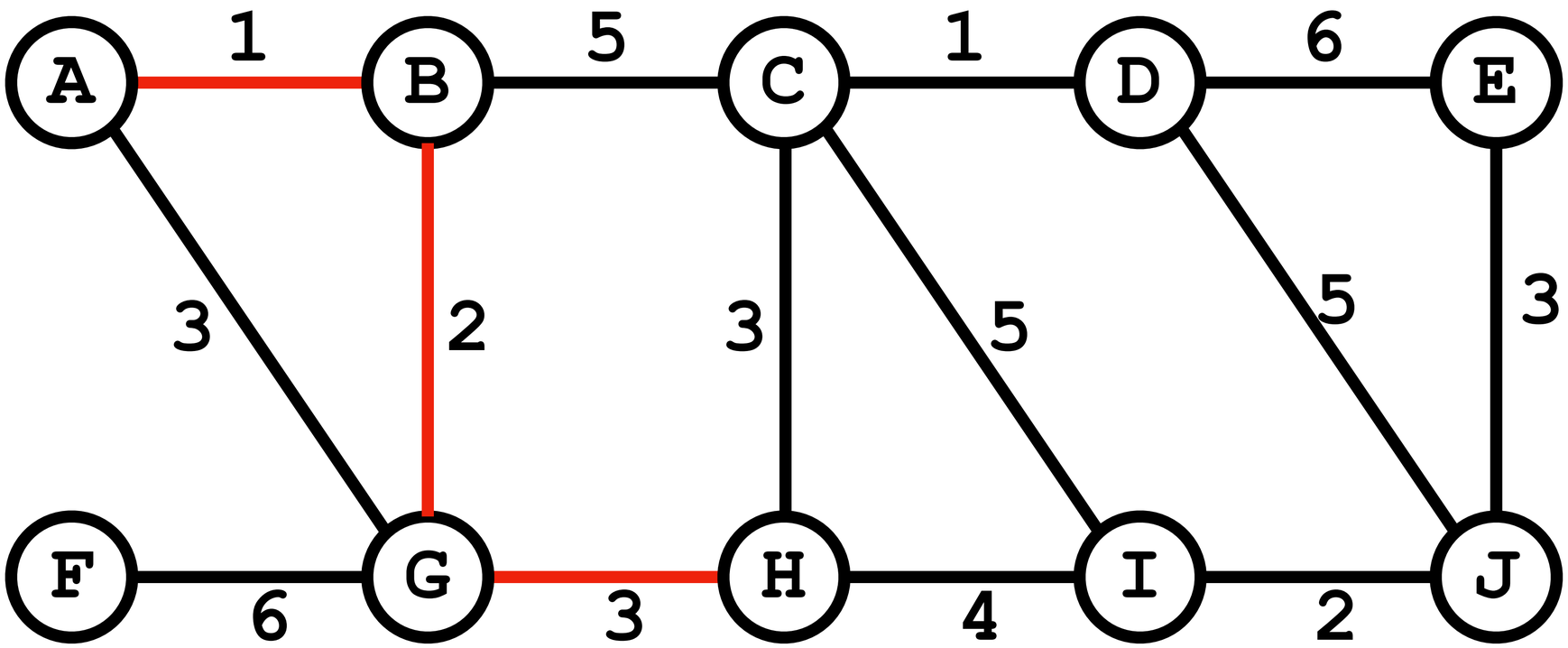
Graphs #2



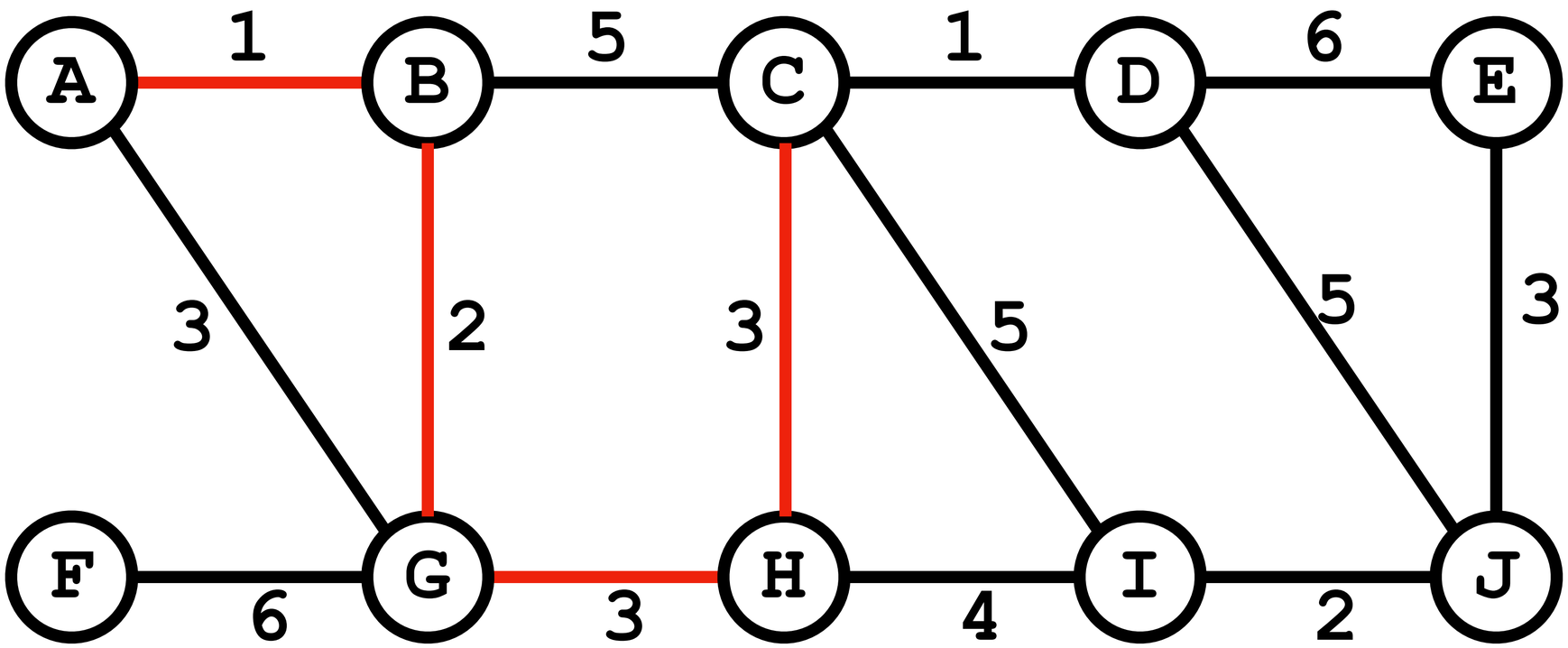
Graphs #2



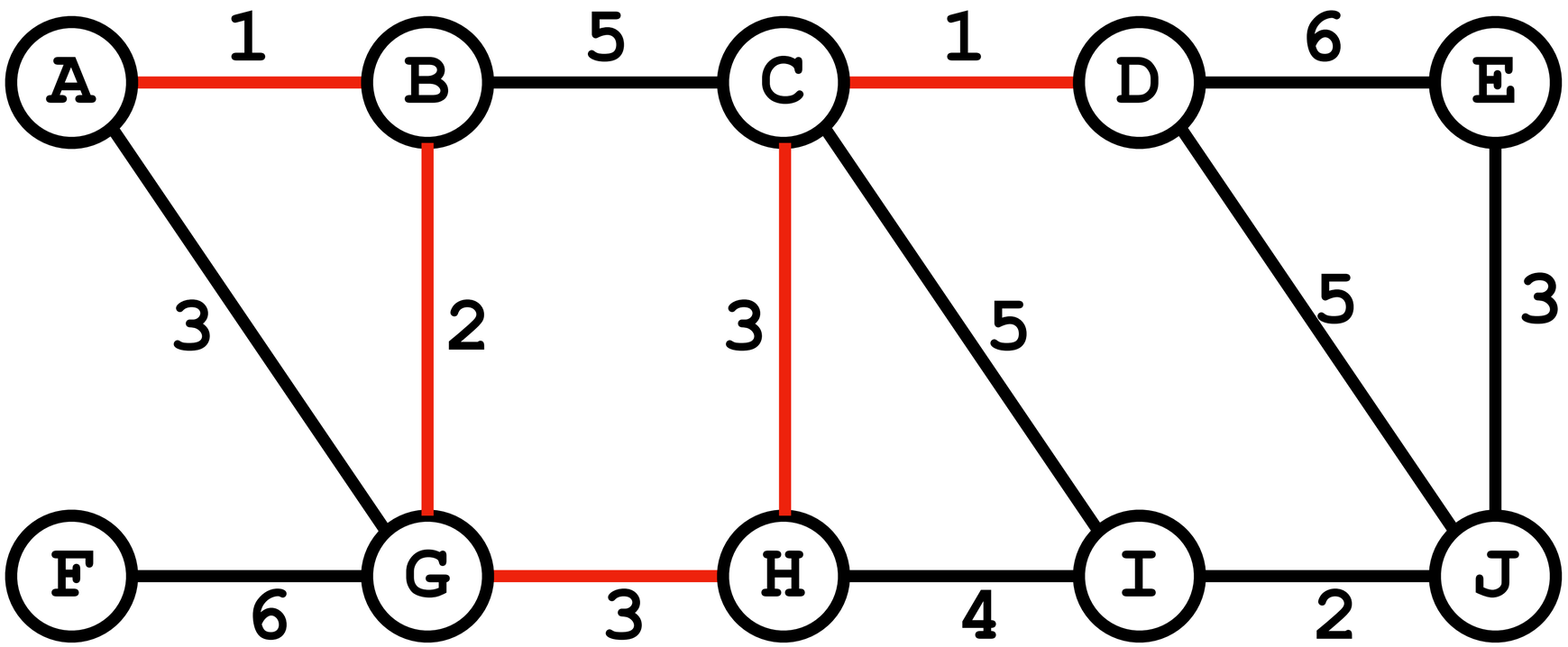
Graphs #2



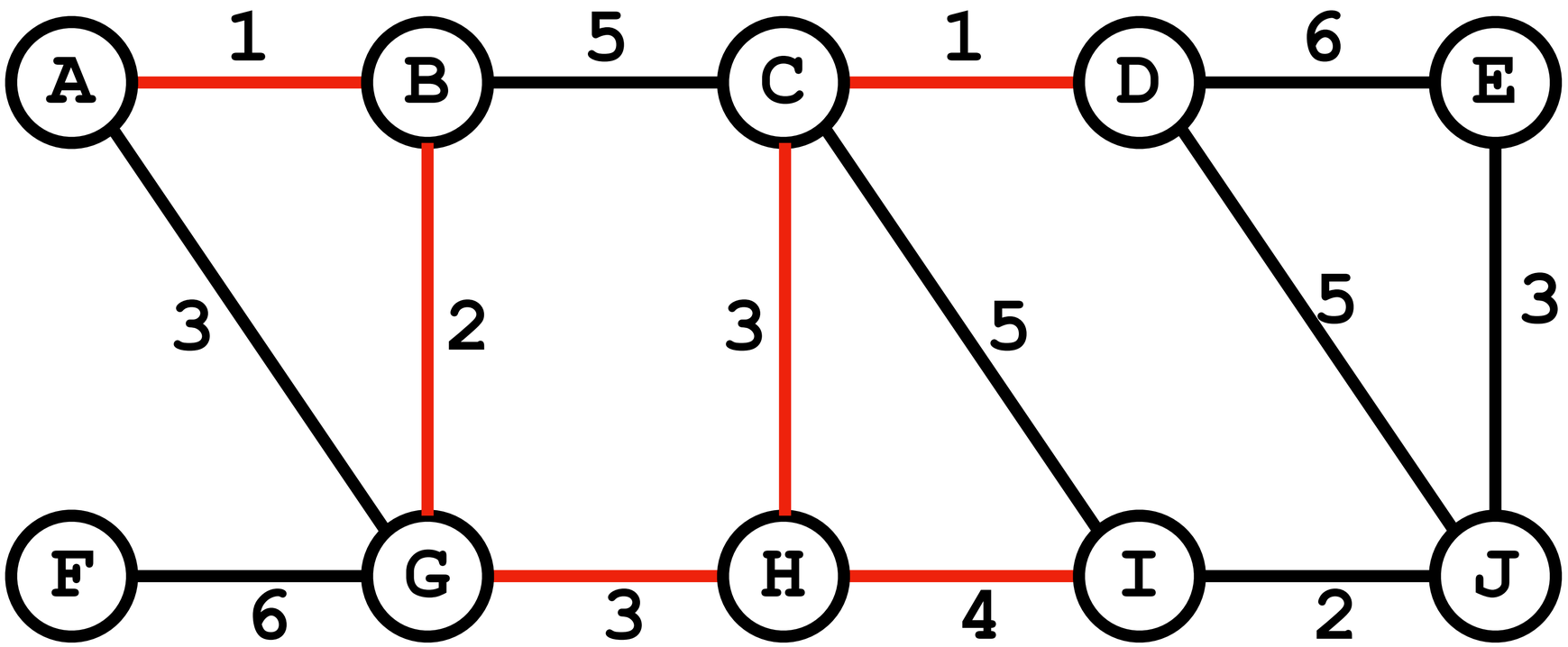
Graphs #2



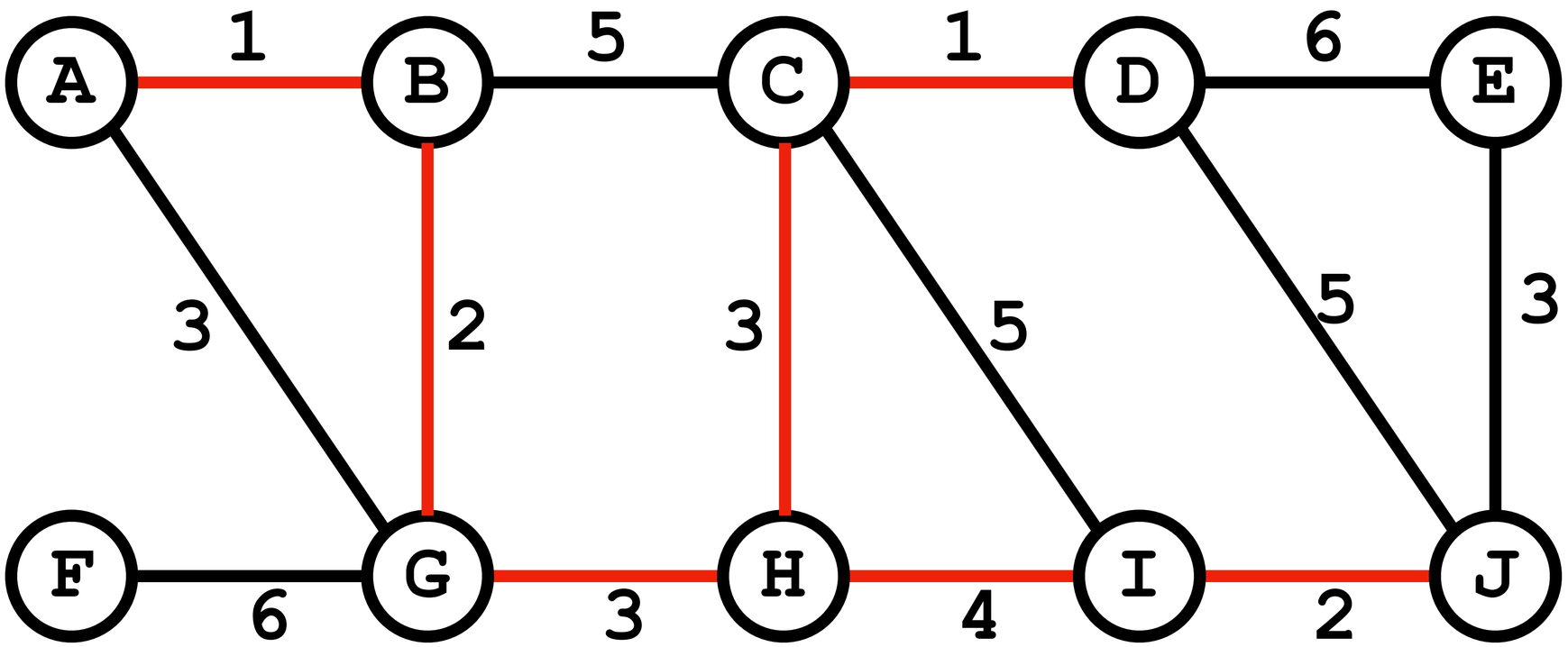
Graphs #2



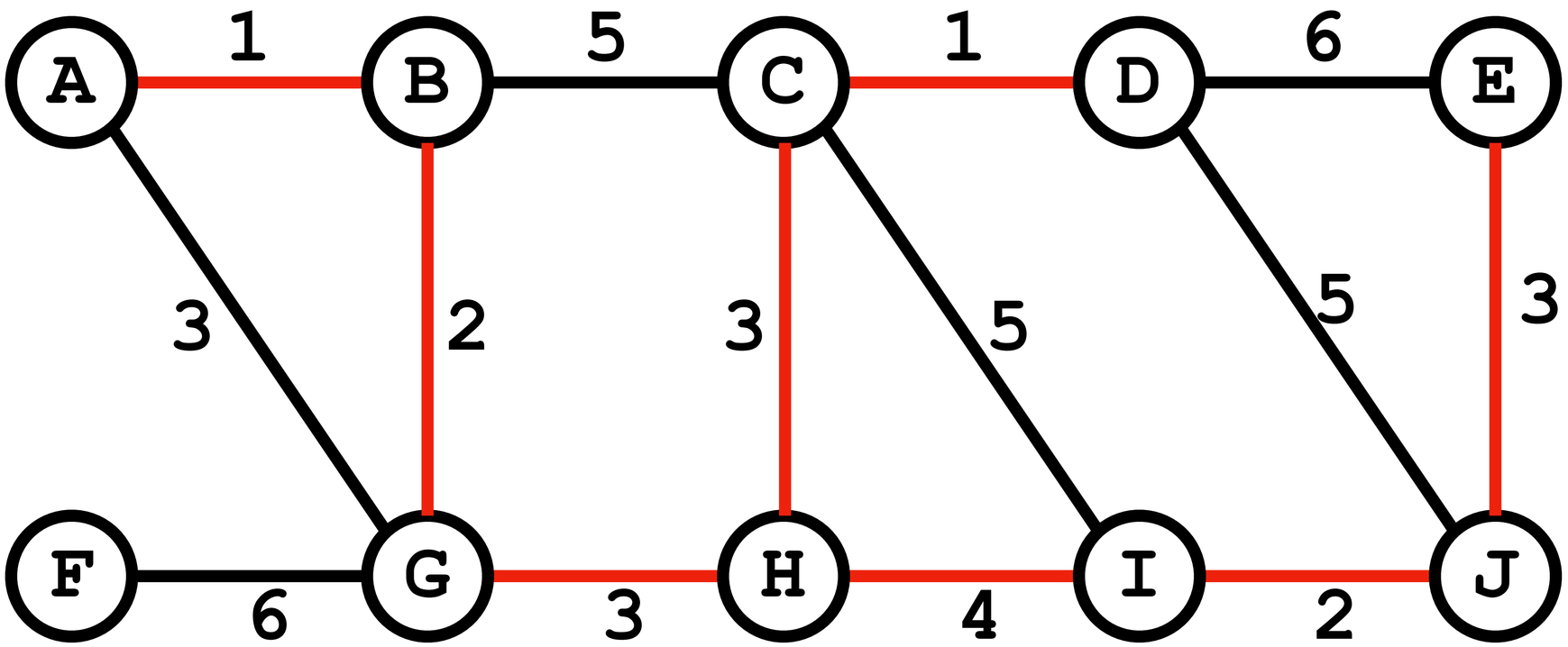
Graphs #2



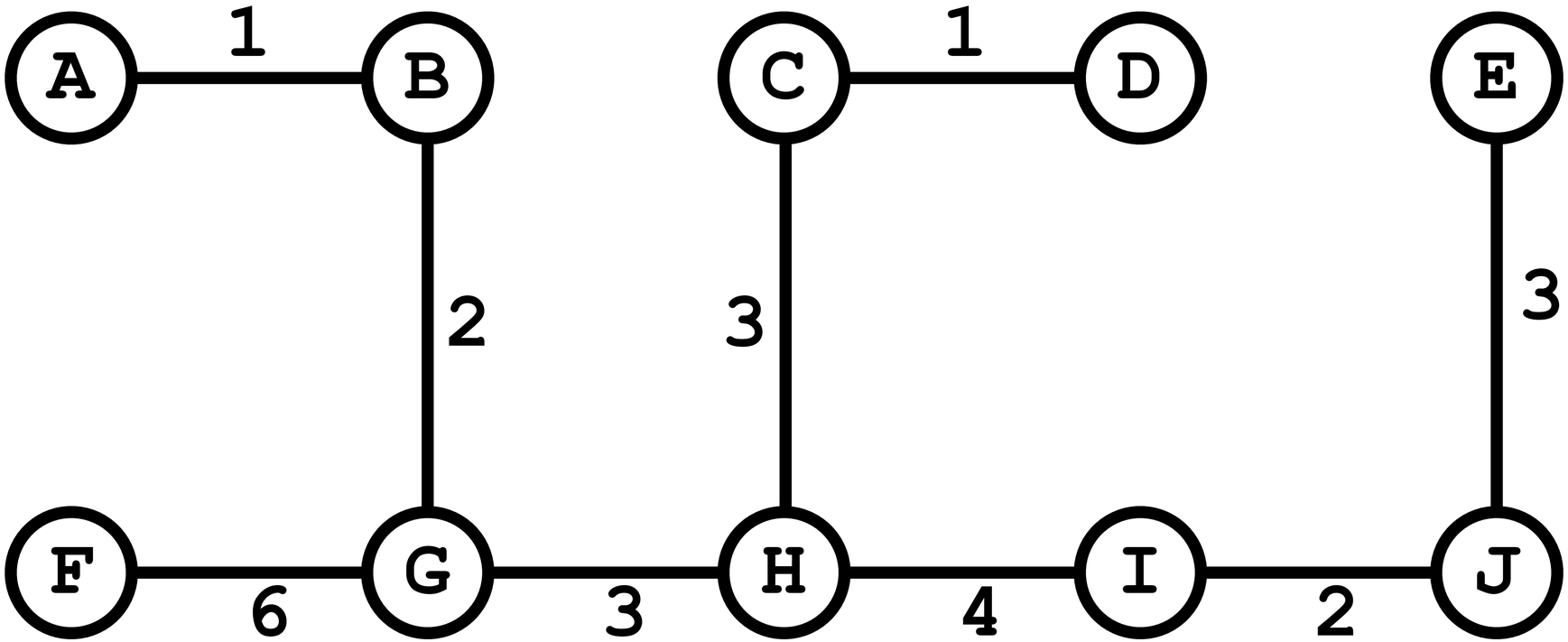
Graphs #2



Graphs #2

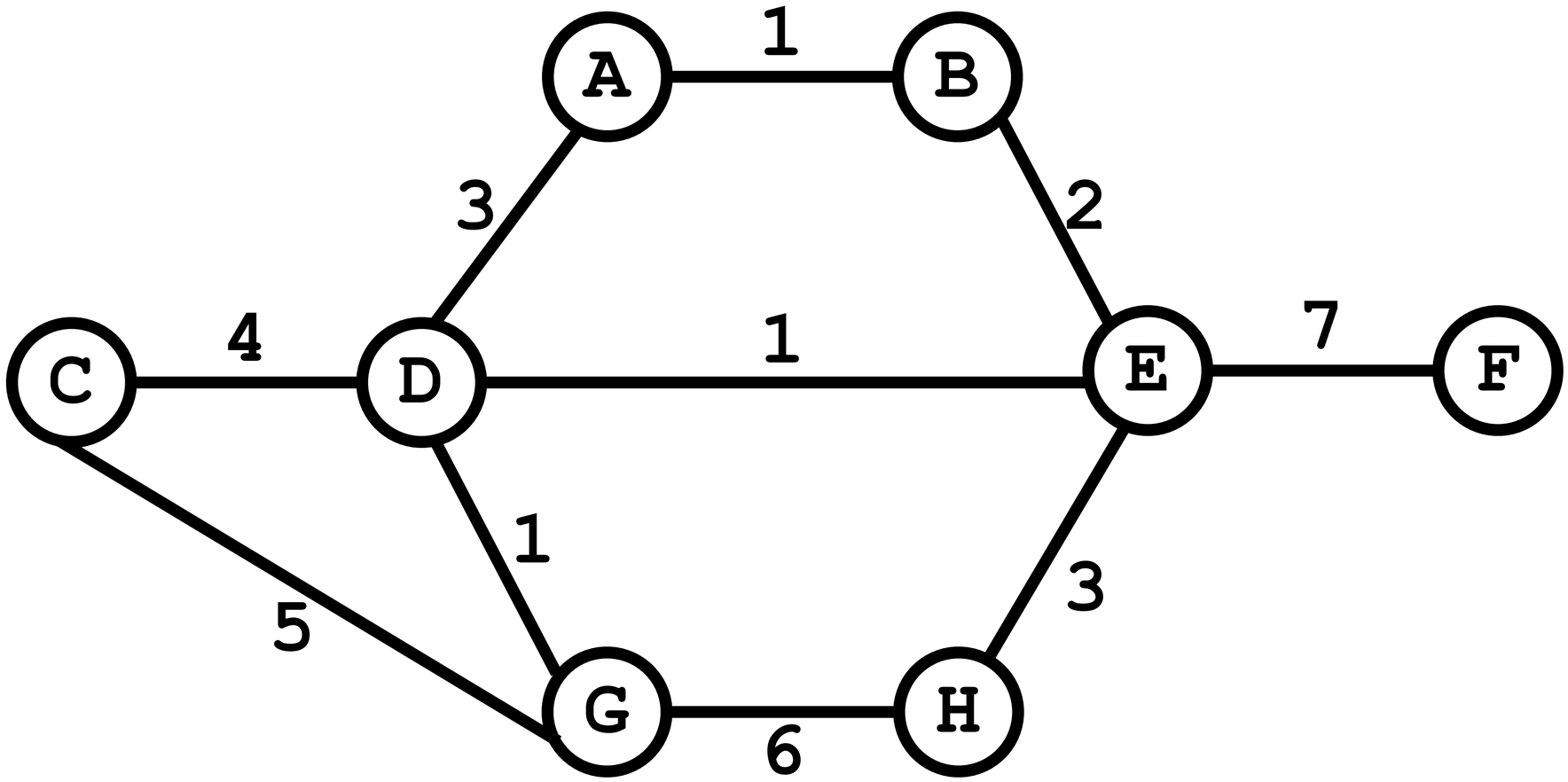


Graphs #2

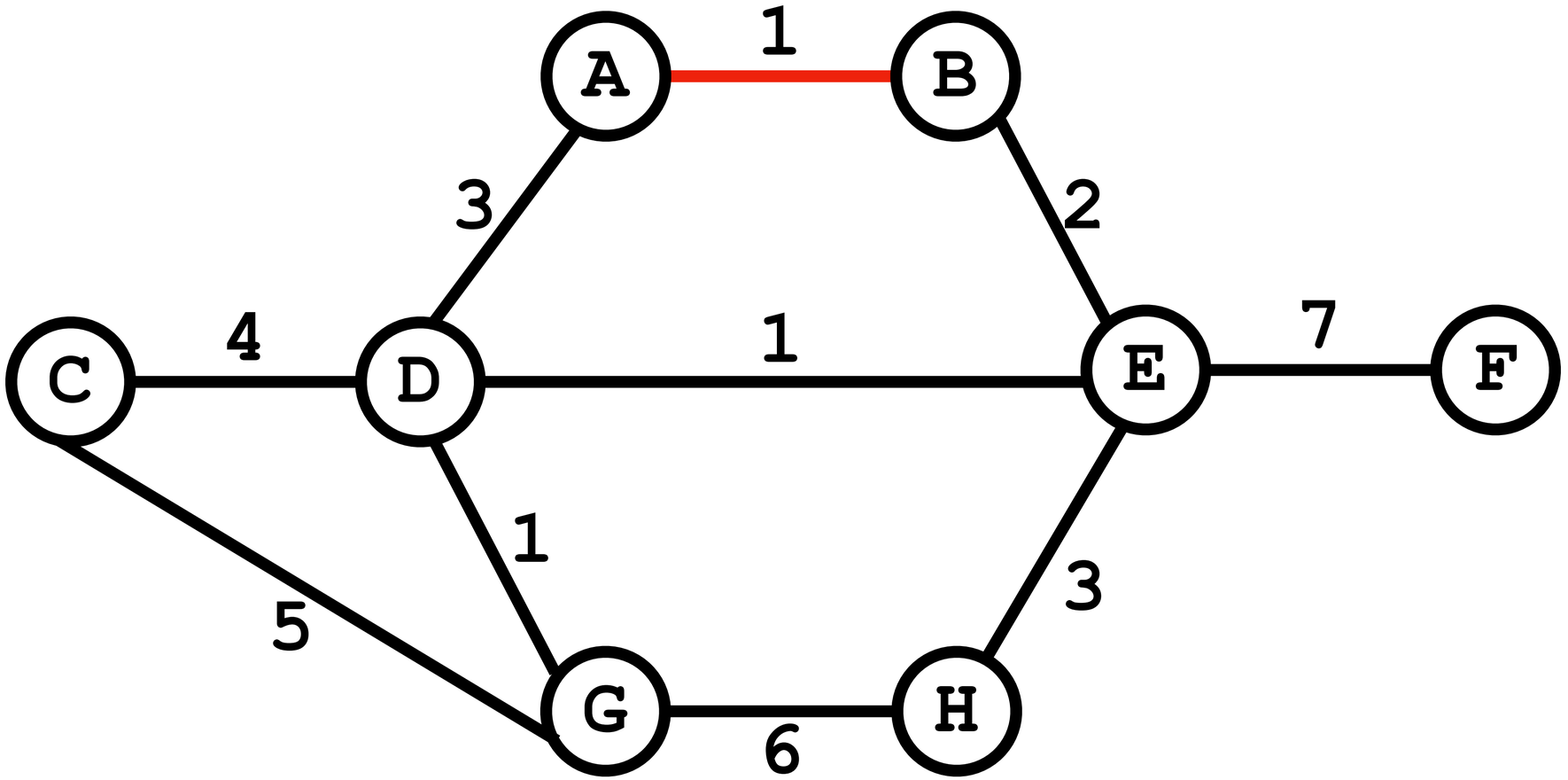


Graphs #3

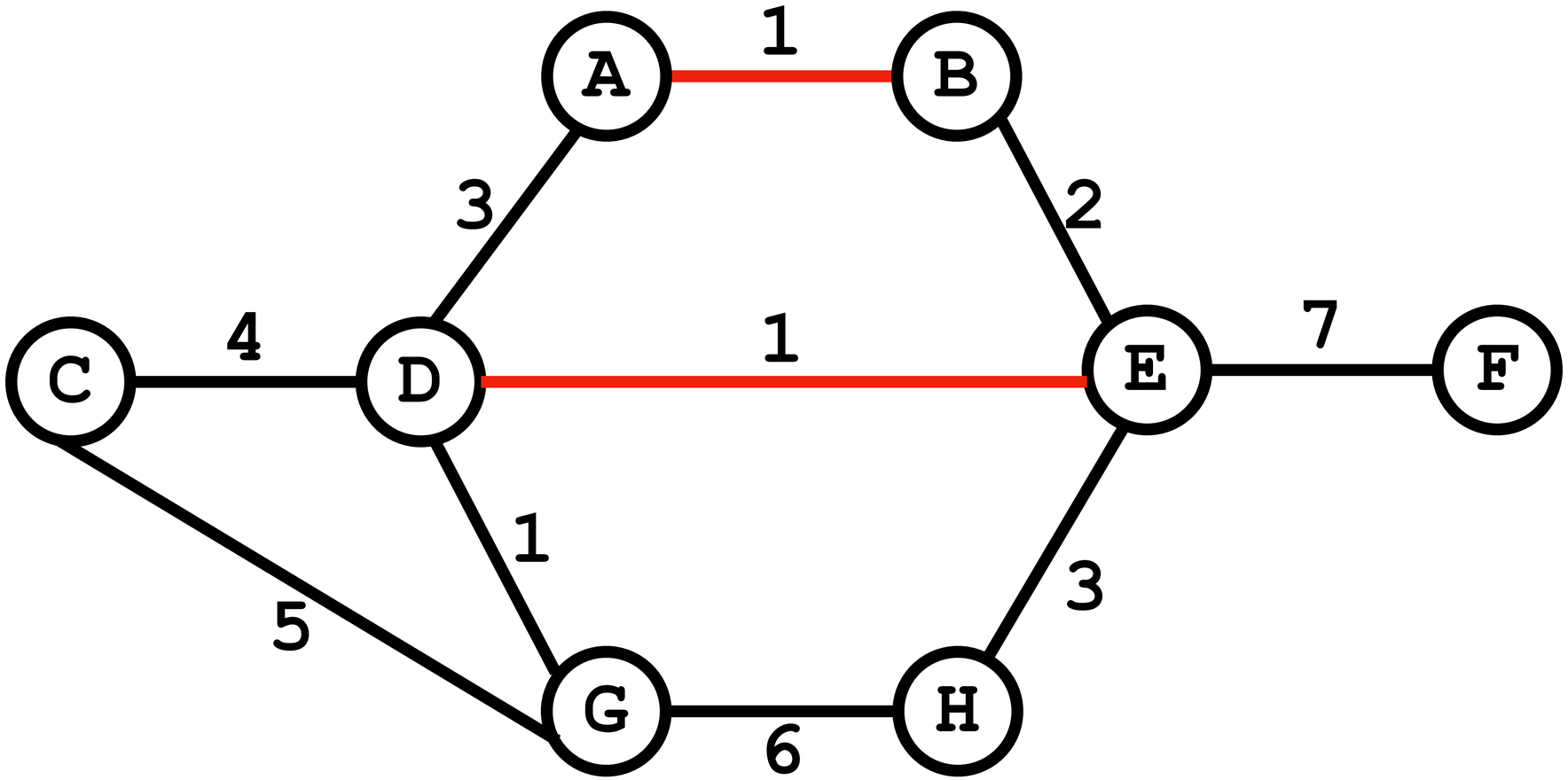
Draw the MST using Kruskal's algorithm.



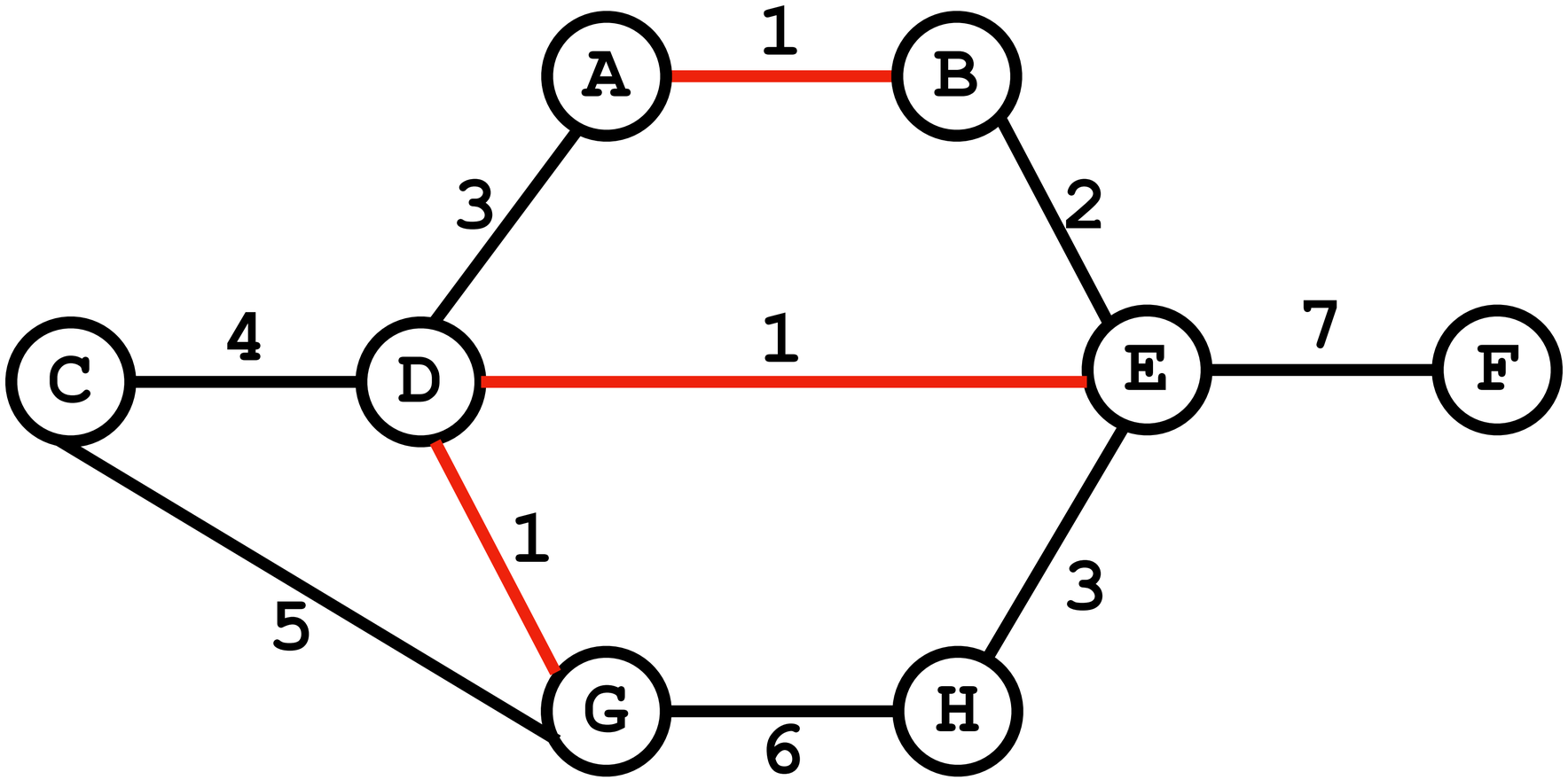
Graphs #3



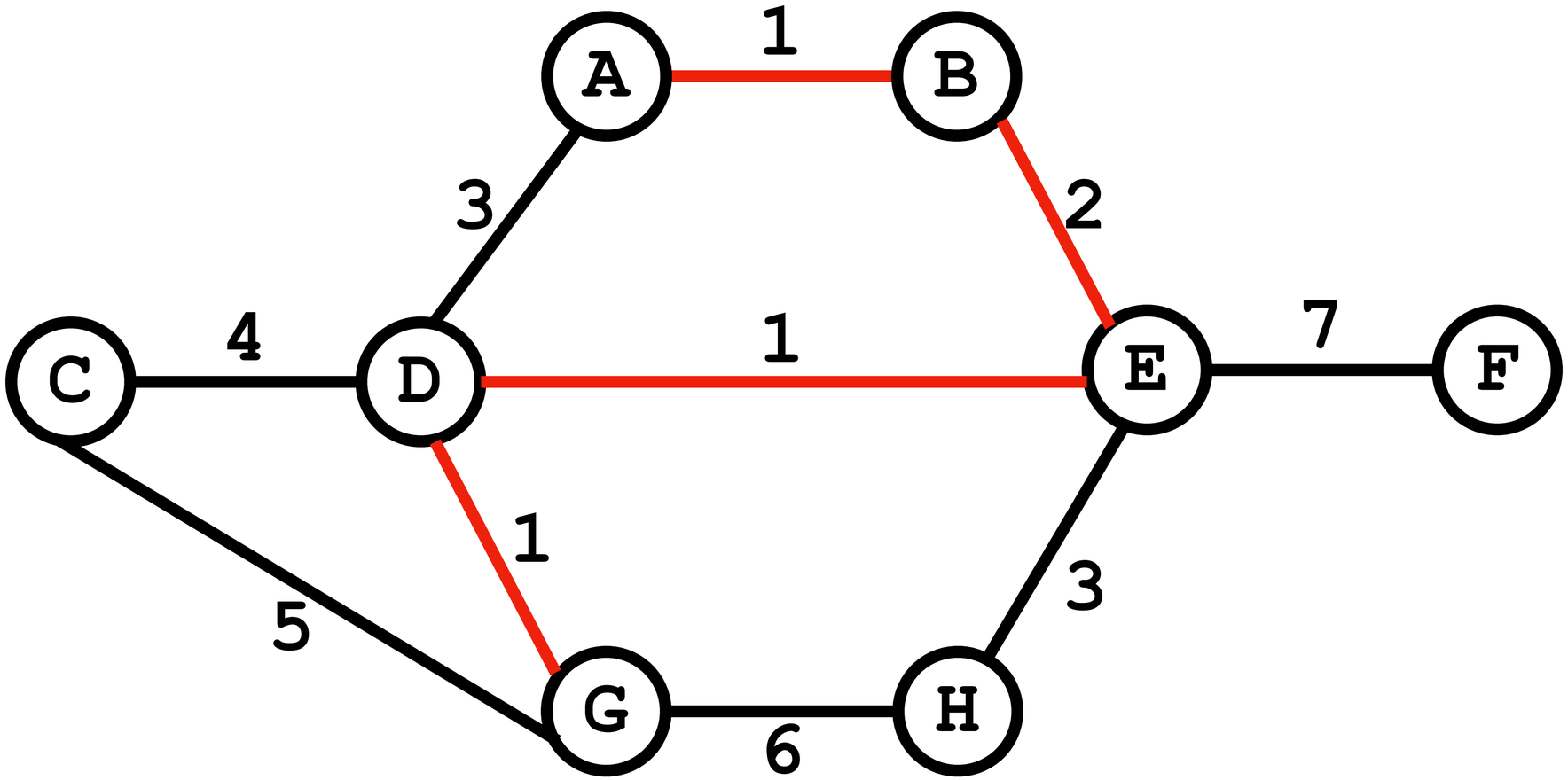
Graphs #3



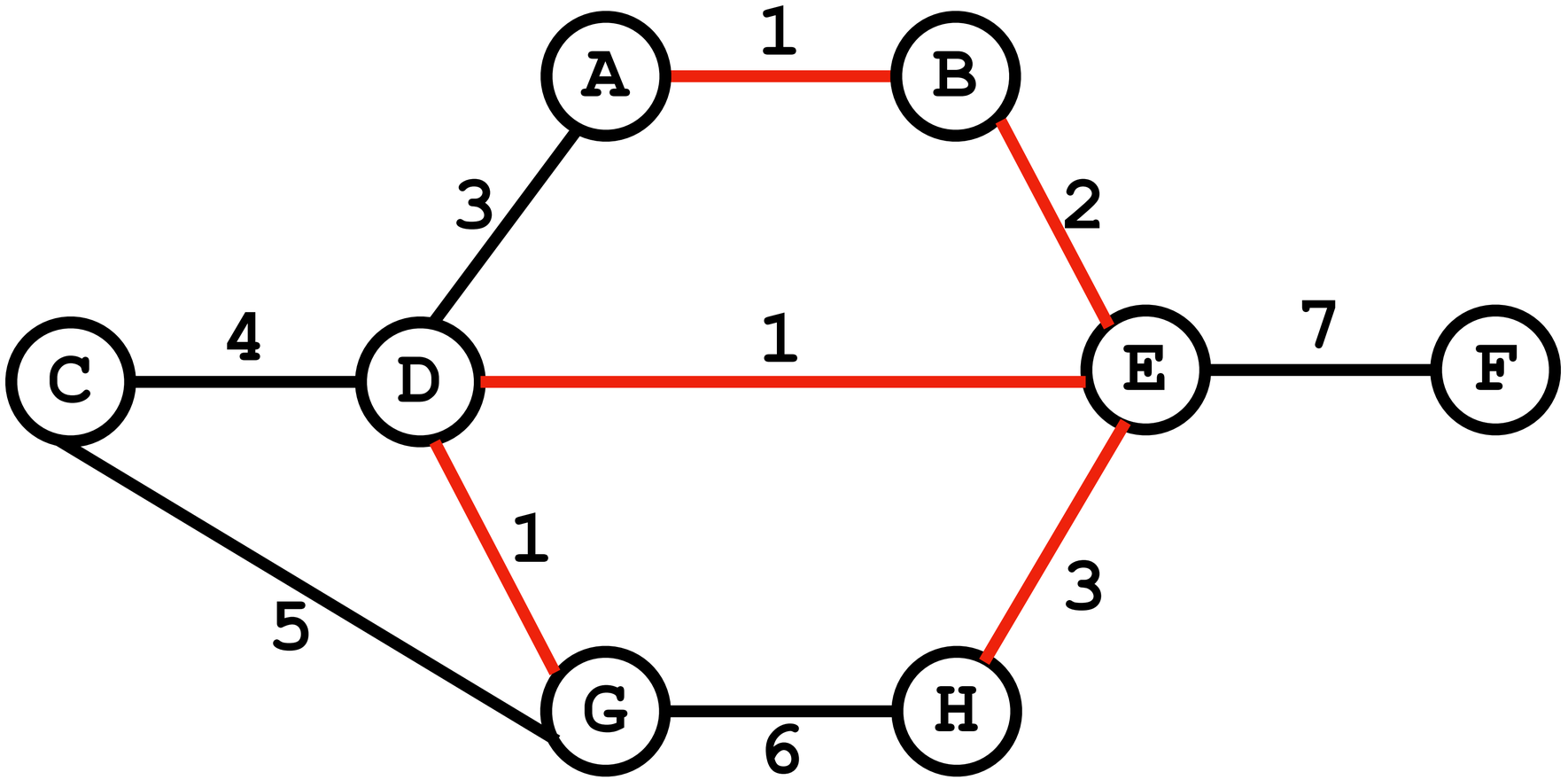
Graphs #3



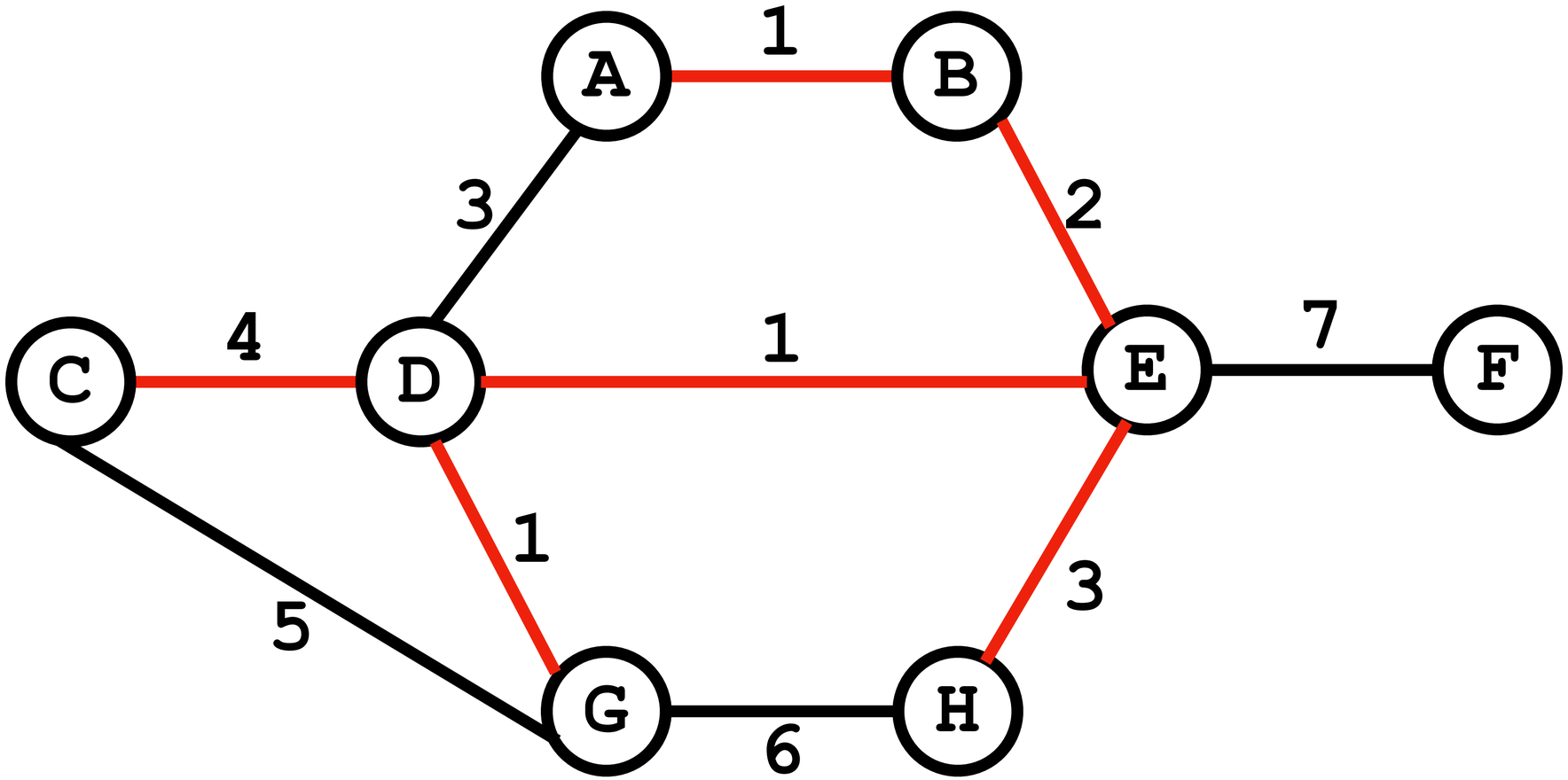
Graphs #3



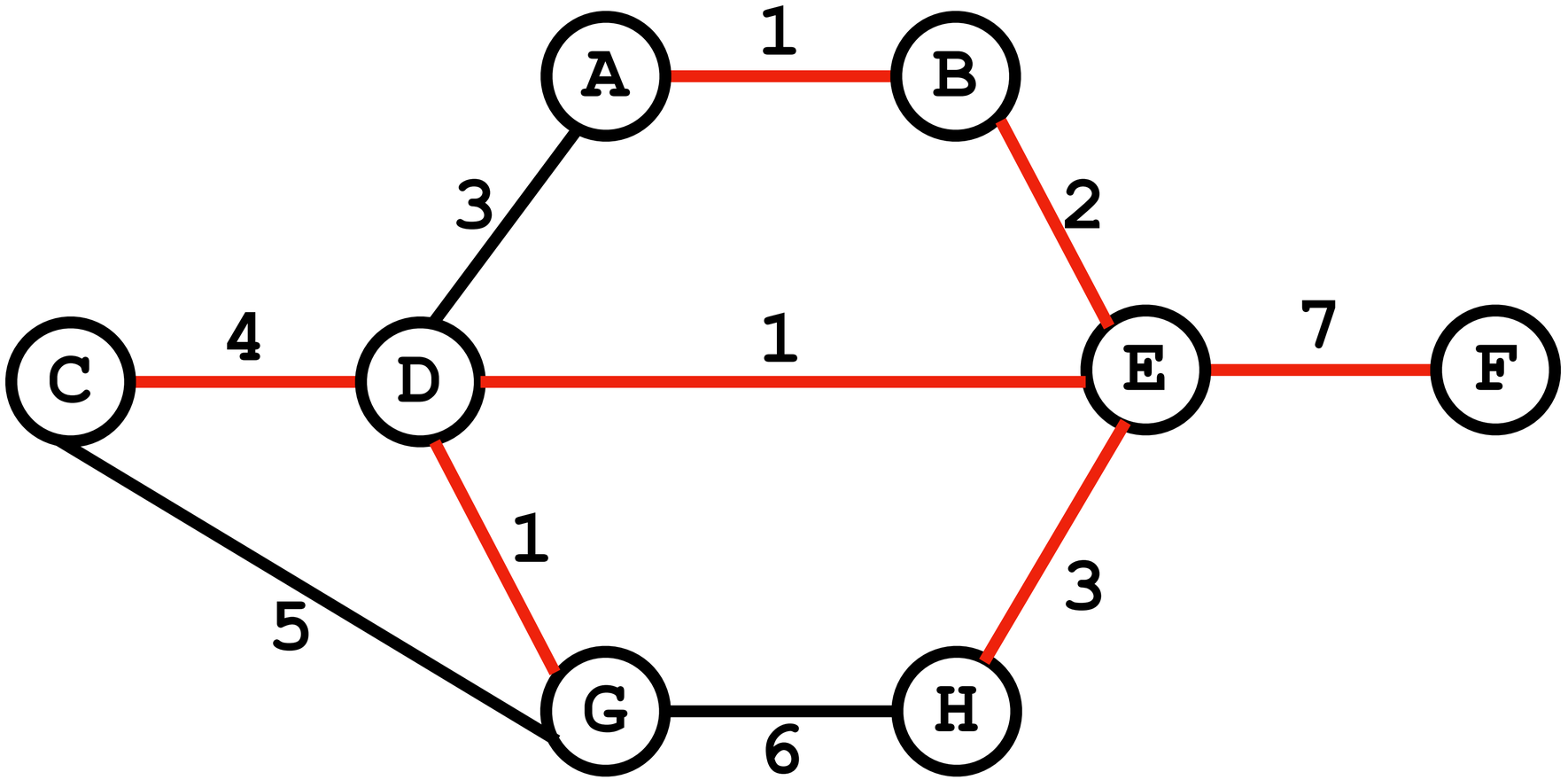
Graphs #3



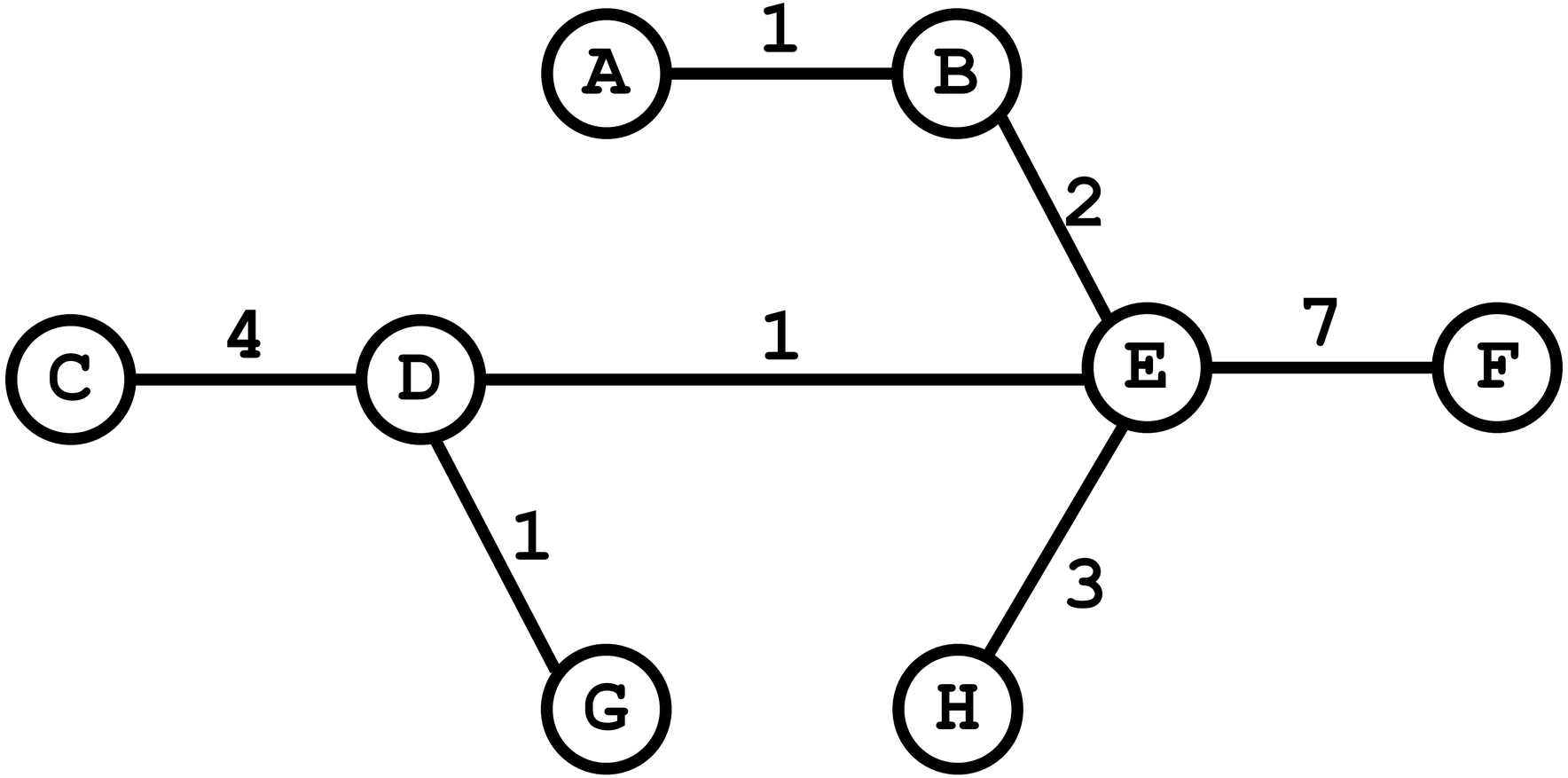
Graphs #3



Graphs #3

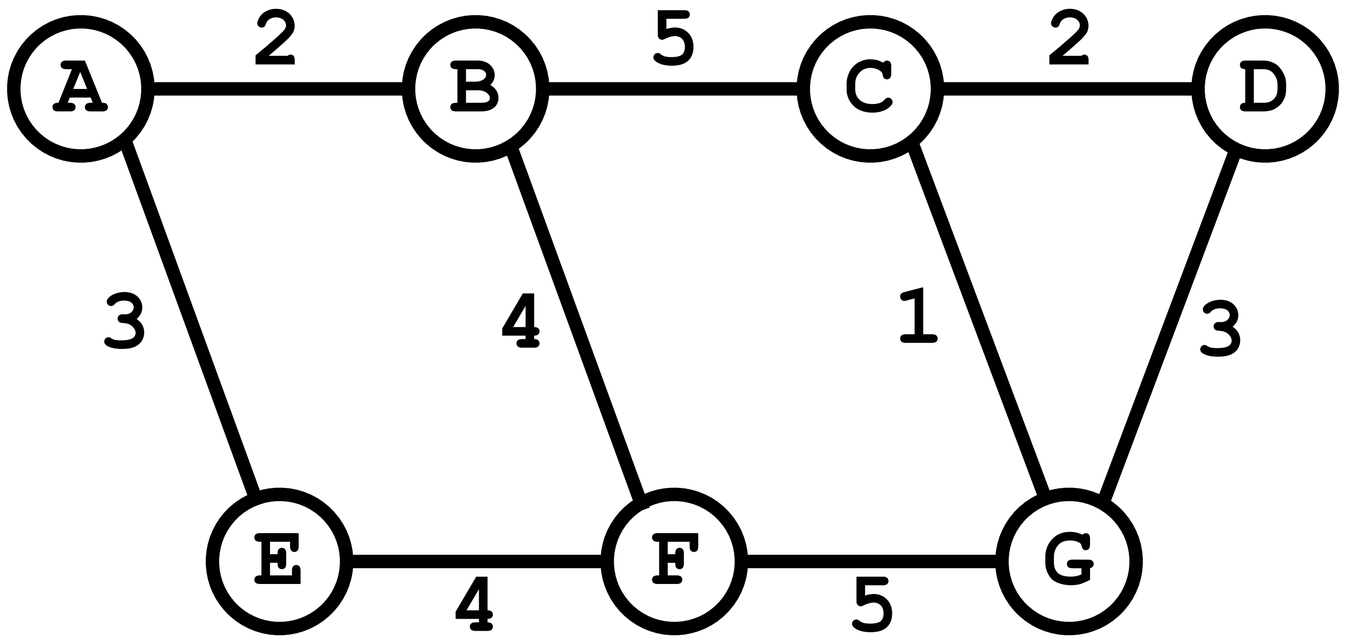


Graphs #3

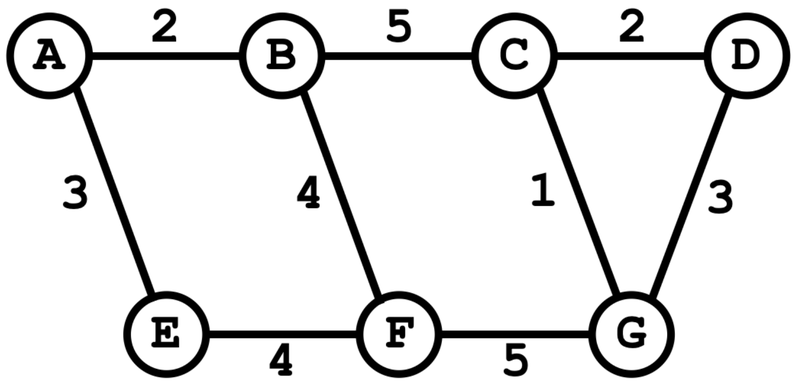


Graphs #4

Given the graph below, draw the shortest path tree from vertex A using Dijkstras.

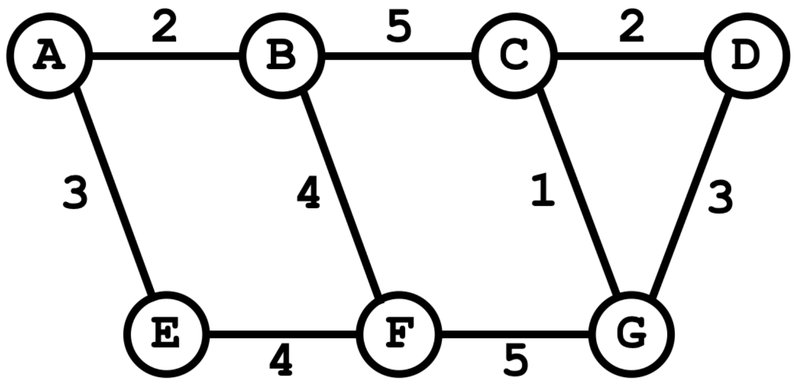


Graphs #4



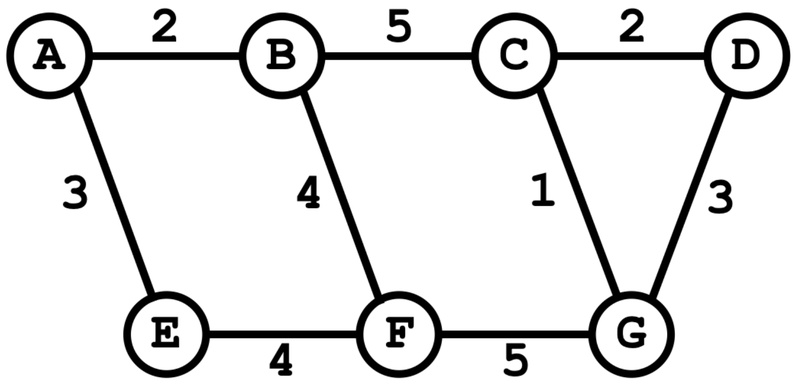
A	B	C	D	E	F	G
0	∞	∞	∞	∞	∞	∞

Graphs #4



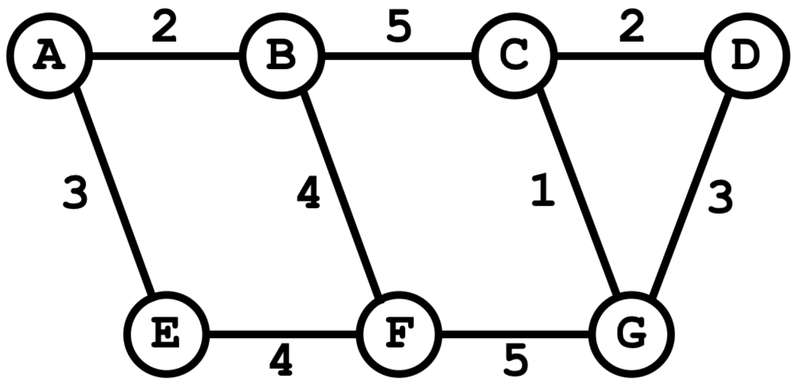
A	B	C	D	E	F	G
0	∞	∞	∞	∞	∞	∞

Graphs #4



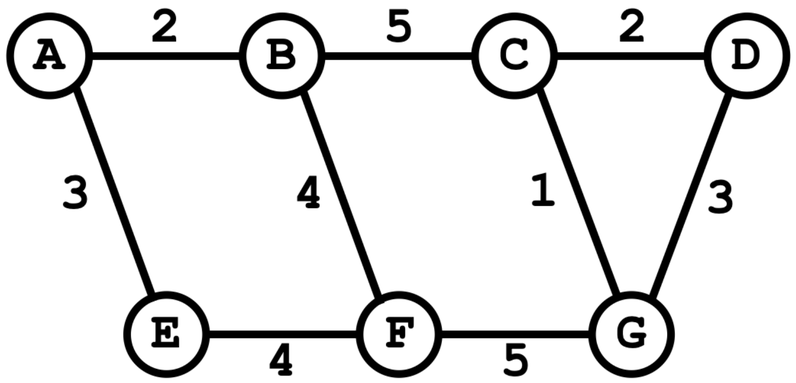
A	B	C	D	E	F	G
0	∞	∞	∞	∞	∞	∞
	2	∞	∞	3	∞	∞

Graphs #4



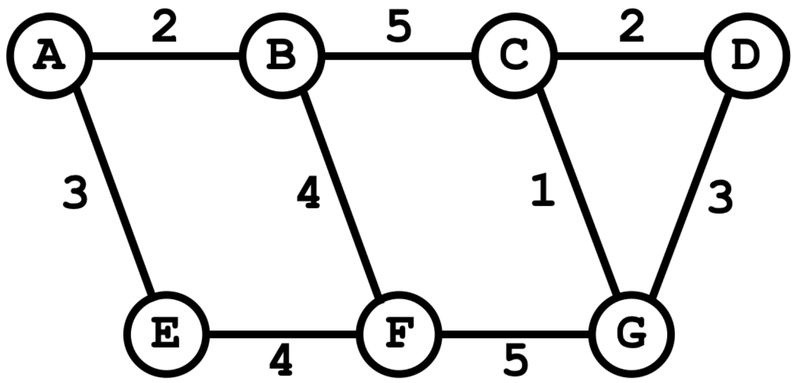
A	B	C	D	E	F	G
0	∞	∞	∞	∞	∞	∞
	2	∞	∞	3	∞	∞
		7	∞	3	6	∞

Graphs #4



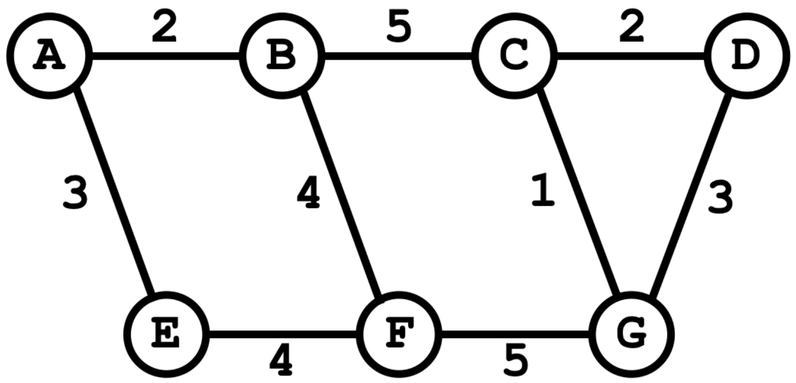
A	B	C	D	E	F	G
0	∞	∞	∞	∞	∞	∞
	2	∞	∞	3	∞	∞
		7	∞	3	6	∞
		7	∞		6	∞

Graphs #4



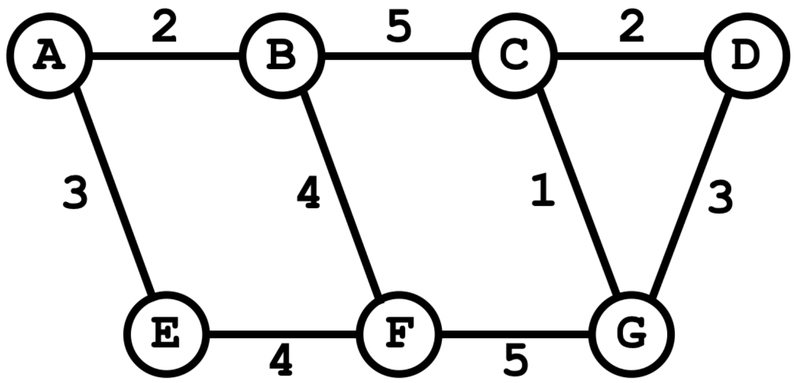
A	B	C	D	E	F	G
0	∞	∞	∞	∞	∞	∞
	2	∞	∞	3	∞	∞
		7	∞	3	6	∞
		7	∞		6	∞
		7	∞			11

Graphs #4



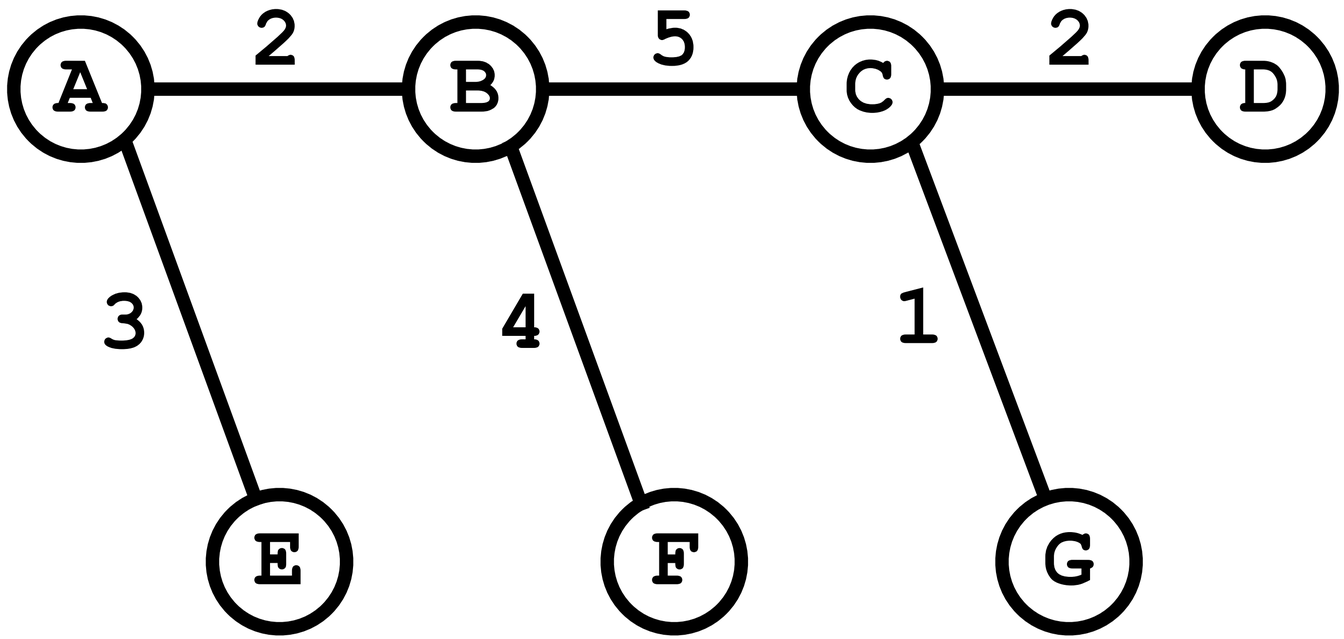
A	B	C	D	E	F	G
0	∞	∞	∞	∞	∞	∞
	2	∞	∞	3	∞	∞
		7	∞	3	6	∞
		7	∞		6	∞
		7	∞			11
			9			8

Graphs #4



A	B	C	D	E	F	G
0	∞	∞	∞	∞	∞	∞
	2	∞	∞	3	∞	∞
		7	∞	3	6	∞
		7	∞		6	∞
		7	∞			11
			9			8
			9			

Graphs #4

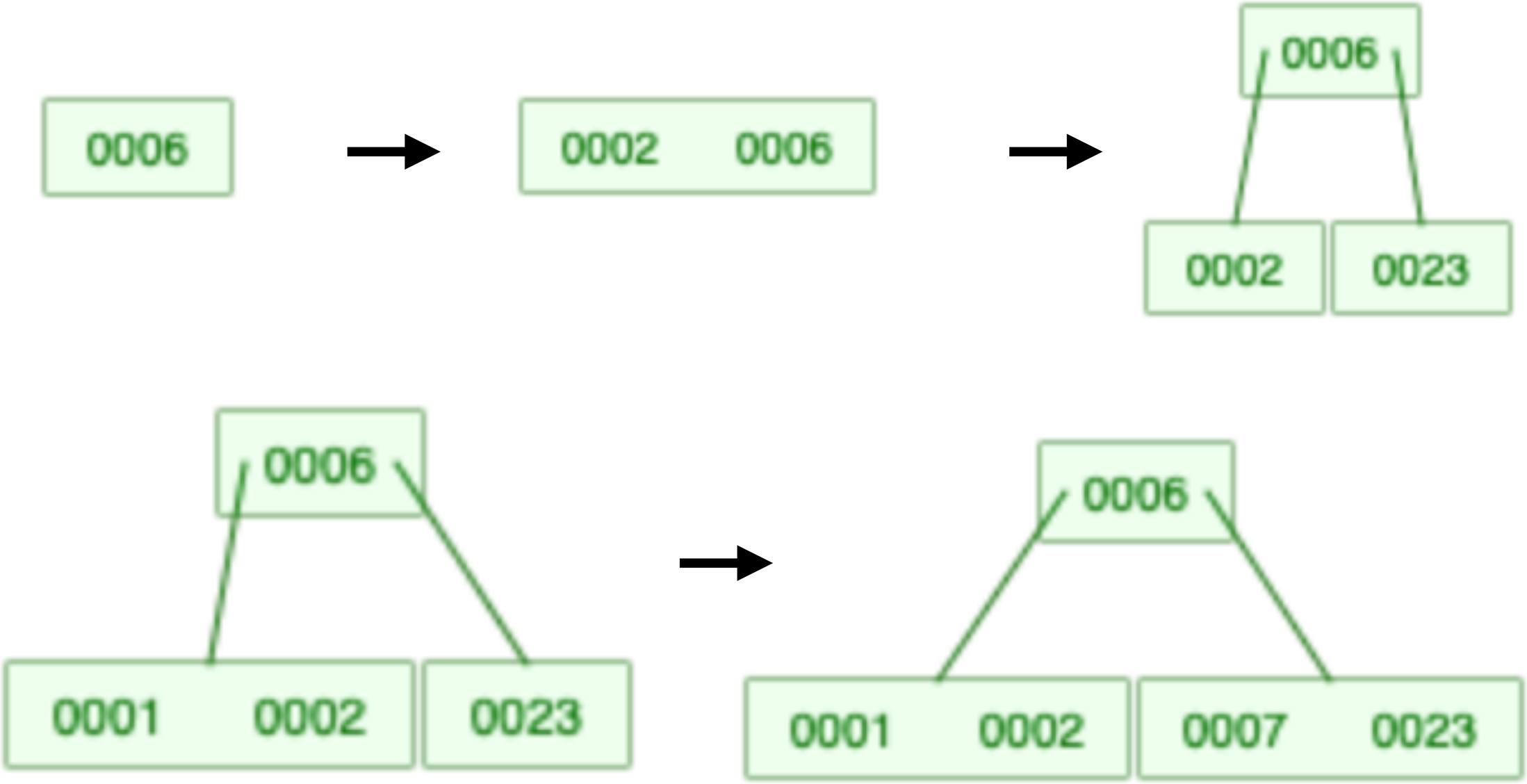


B-Tree #1

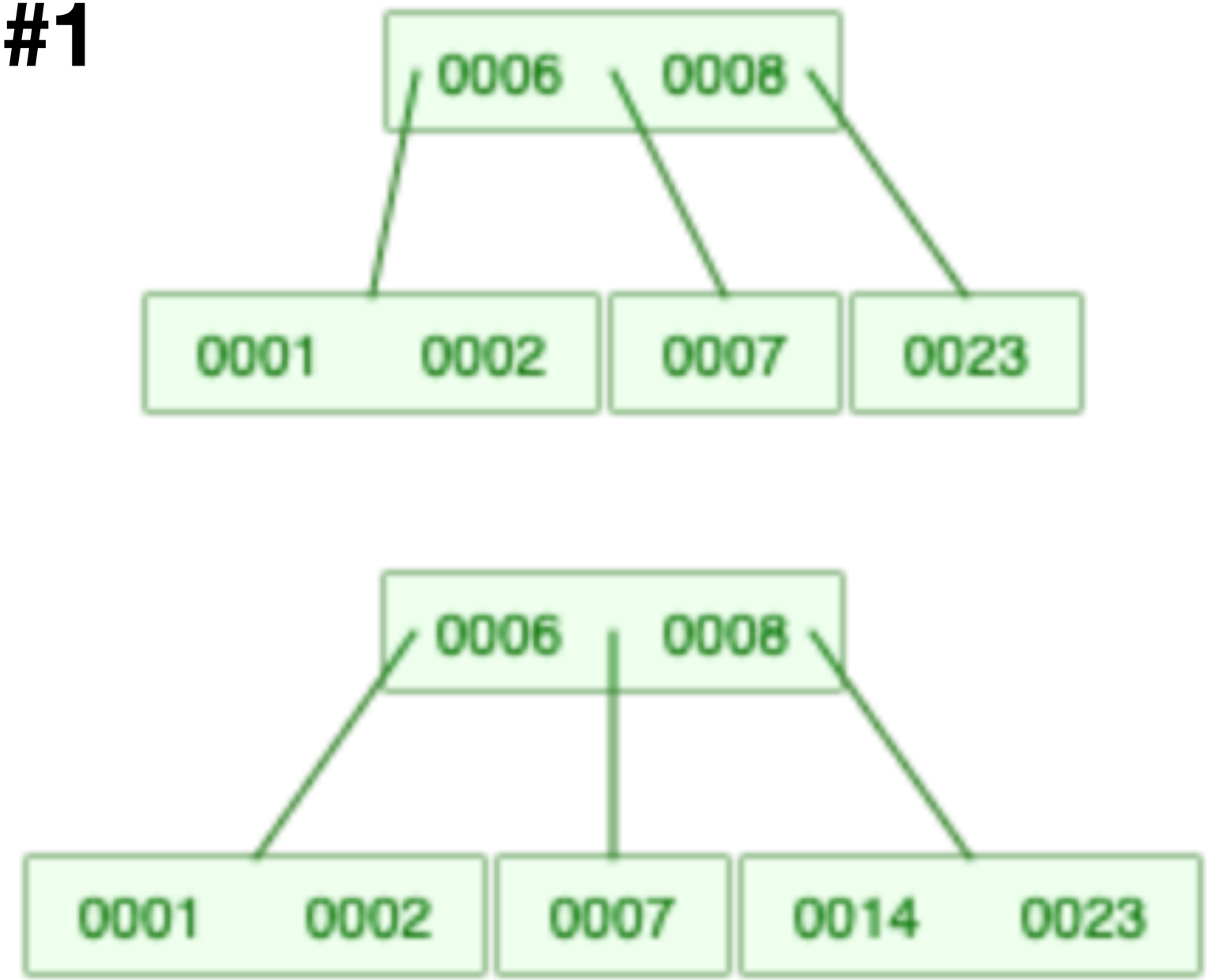
Insert the following values into a B-Tree of degree 3 in the order in which they appear.

6, 2, 23, 1, 7, 8, 14, 55

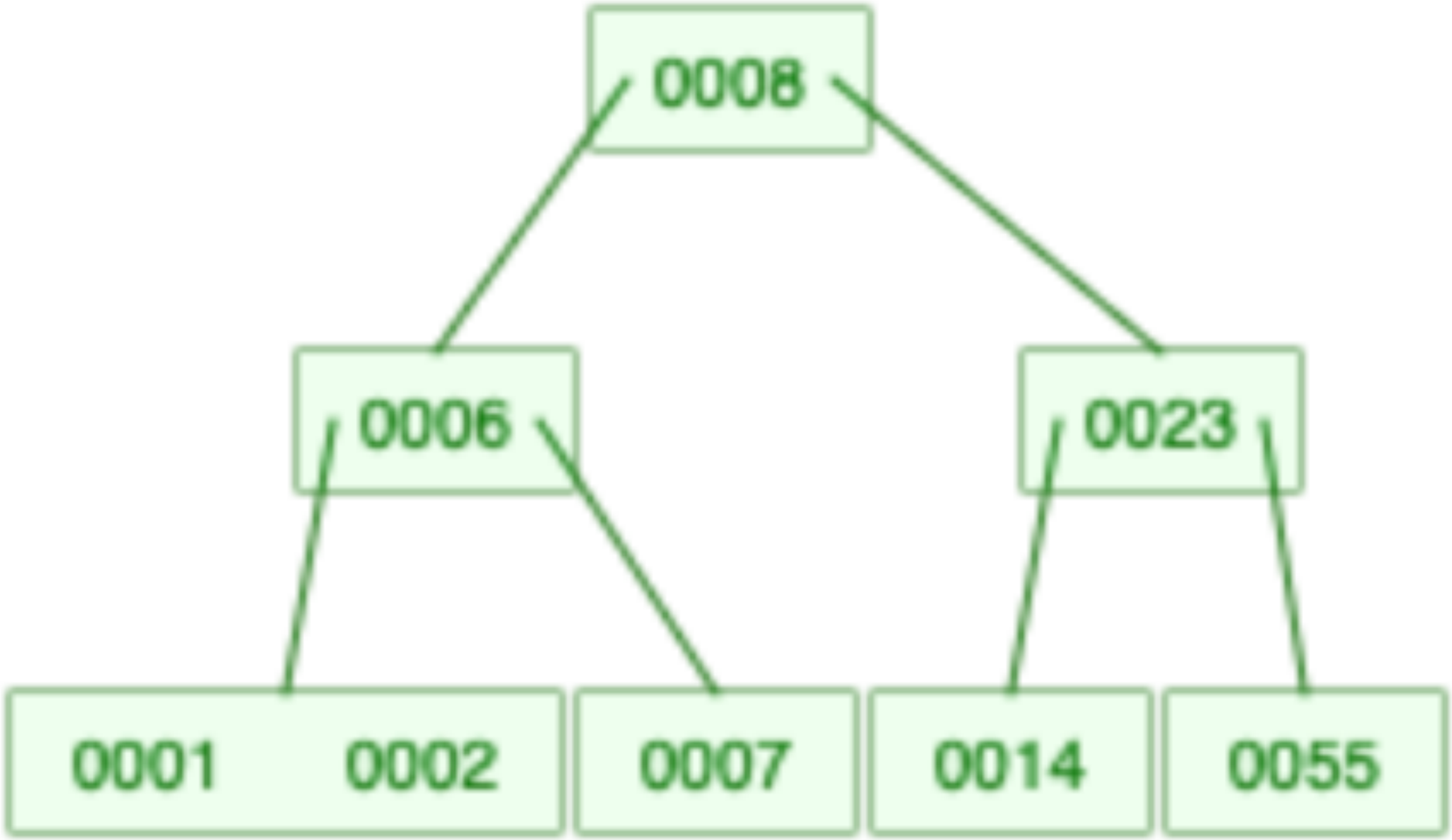
B-Tree #1


0006
0002 0006
0006
0002 0023
0006
0001 0002 0023
0006
0001 0002 0007 0023

B-Tree #1


0006
0008
0001
0002
0007
0023
0006
0008
0001
0002
0007
0014
0023

B-Tree #1



Final Exam Overview

Section	Question	Points Possible	Question Type
BST	I	6	Coding
	II	7	Coding
	III	7	Coding
	IV	7	Coding
	V	7	Coding
	VI	7	Coding
AVL Tree	I	6	Coding
	II	7	Coding
	III	3	Tracing
Graph	I	3	Tracing
	II	3	Tracing
	III	8	Coding
	IV	7	Tracing
	V	7	Tracing
	VI	7	Tracing
B-Tree	I	8	Tracing
Total	**16**	**100**	