Data Structure Visualizations

reveal.js plugin

Created by Kostas Chatzikokolakis using David Galles' visualization library.

Demo

- Shows many of the provided animations
  - full list is available here
- Each action is shown just once (to keep it short)
- Actions are reveal.js fragments
  - easy to create arbitrary sequences

Navigation

right / **left** : move one **action** at a time.

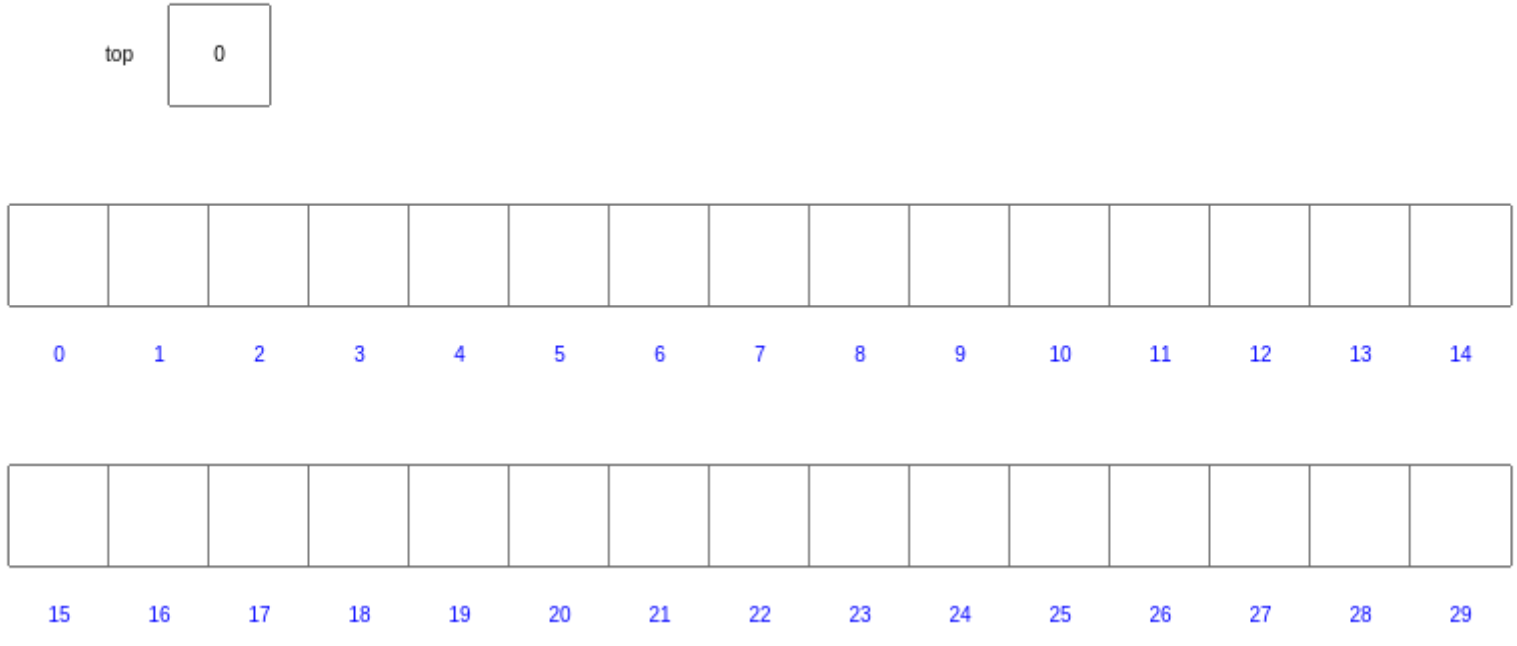
down / **up** : same (vertical slides)

Space,N / **P** : animate one **step** at a time.

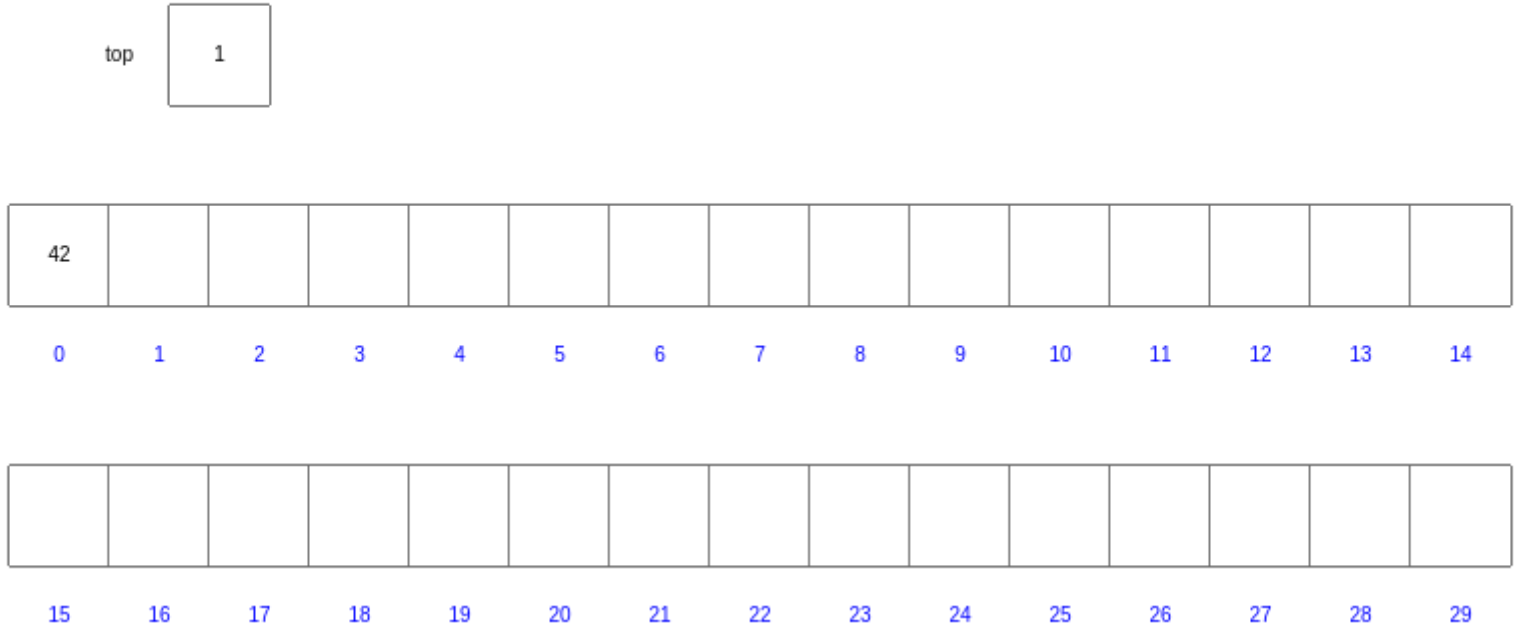
Basics

- Stack: Array Implementation
- Stack: Linked List Implementation
- Queue: Array Implementation
- Queue: Linked List Implementation
- Binary and Linear Search

Stack: Array Implementation

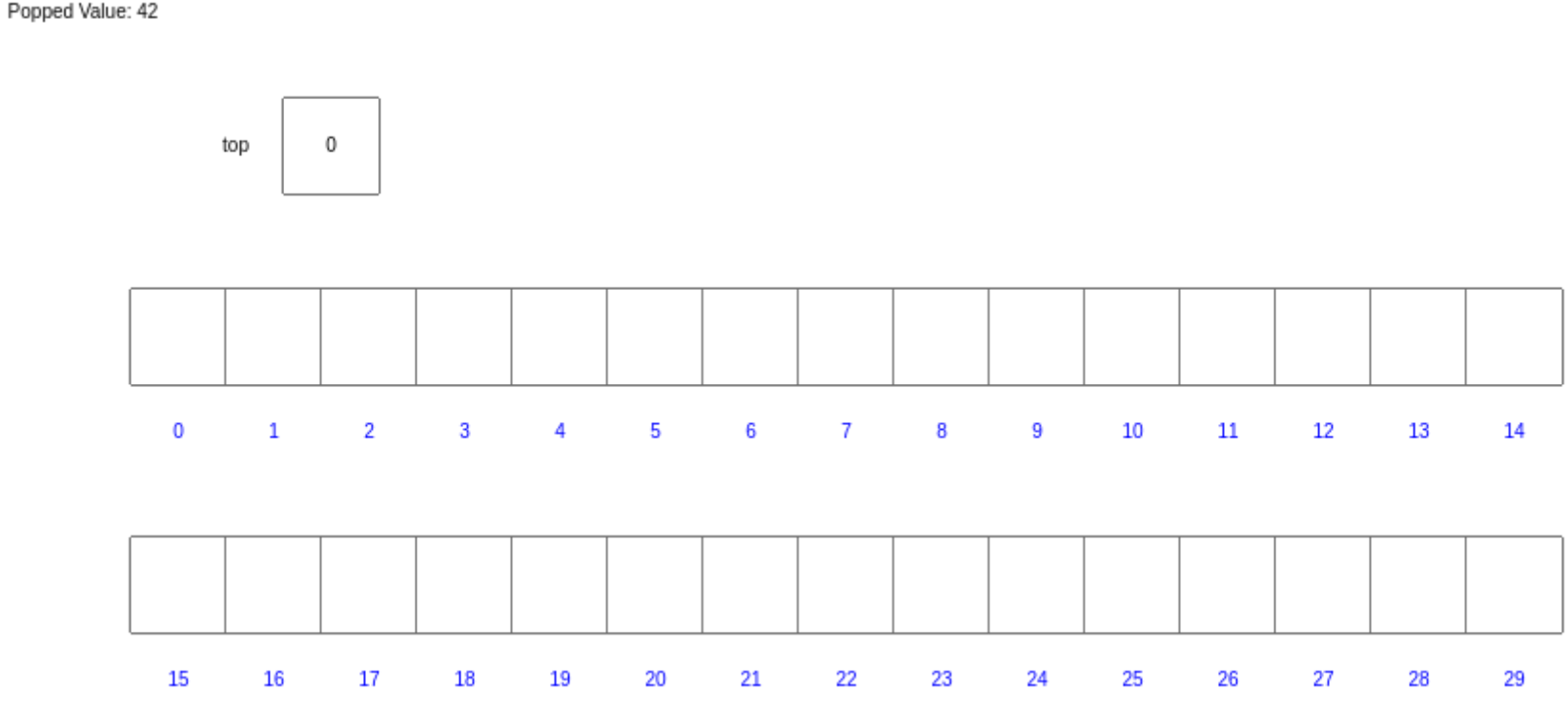


Stack: Array Implementation



Push

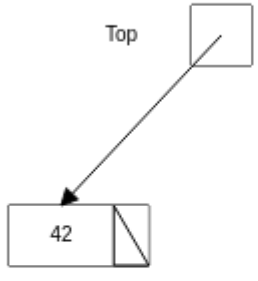
Stack: Array Implementation



Pop

Stack using Linked List

Stack using Linked List



Push

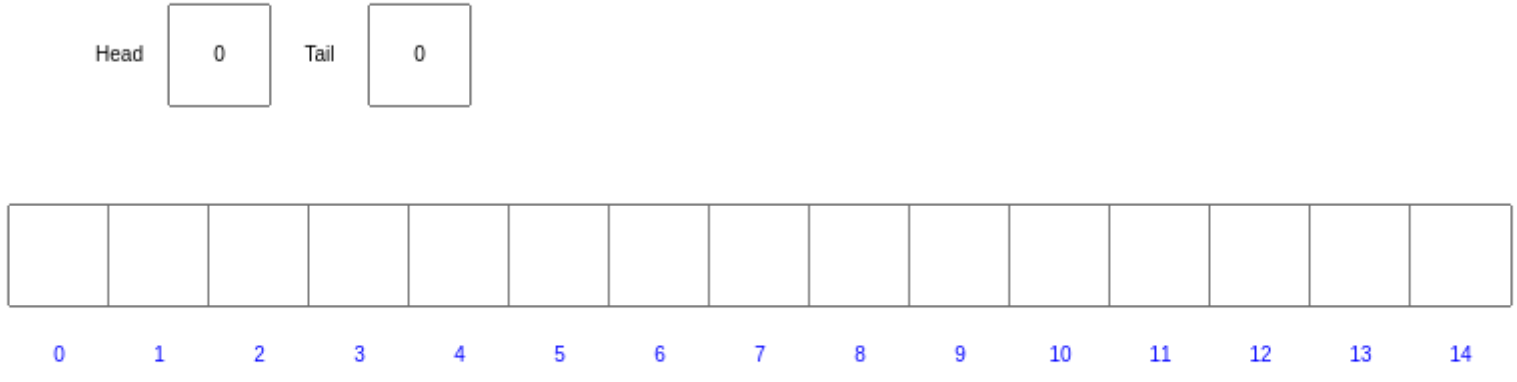
Stack using Linked List

Popped Value: 42

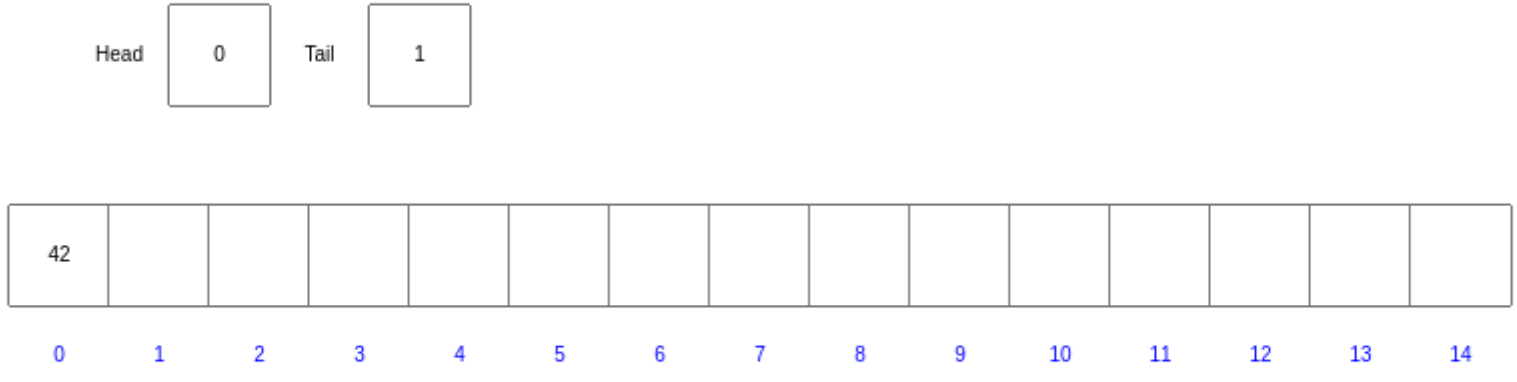
Top 

Pop

Queue using Array

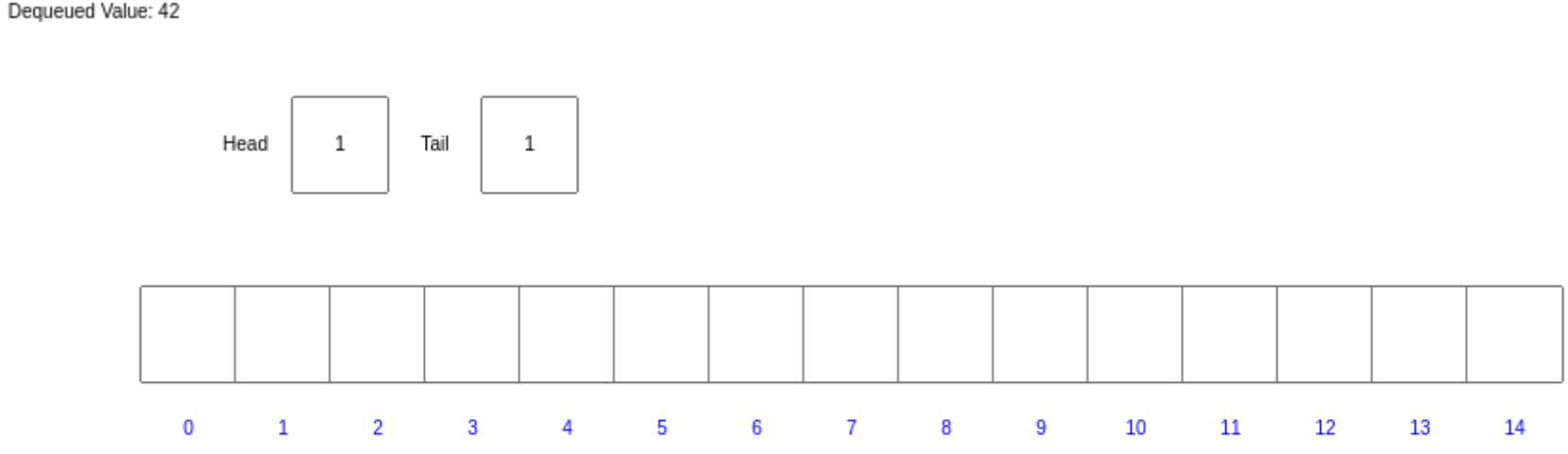


Queue using Array



Enqueue

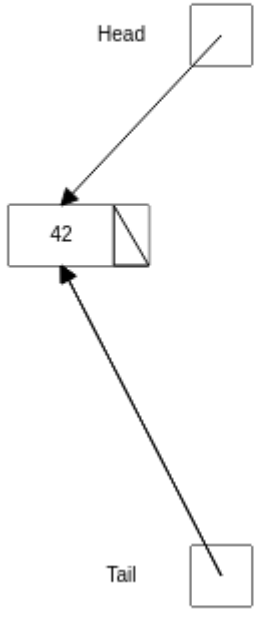
Queue using Array



Dequeue

Queue using Linked List

Queue using Linked List



Enqueue

Queue using Linked List

Dequeued Value: 42

Head 

Tail 

Dequeue

Binary and Linear Search

Seaching For

Result

5	50	50	65	90	119	210	248	270	346	356	387	396	425	434	455
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

519	547	598	604	721	748	748	792	874	876	888	897	900	941	973	976
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Binary and Linear Search

```
def linearSearch(listData, value)
    index = 0
    while (index < len(listData) and listData[index] < value):
        index++;
    if (index >= len(listData or listData[index] != value):
        return -1
    return index
```

Seaching For 346 Result 9 Element found

index 9

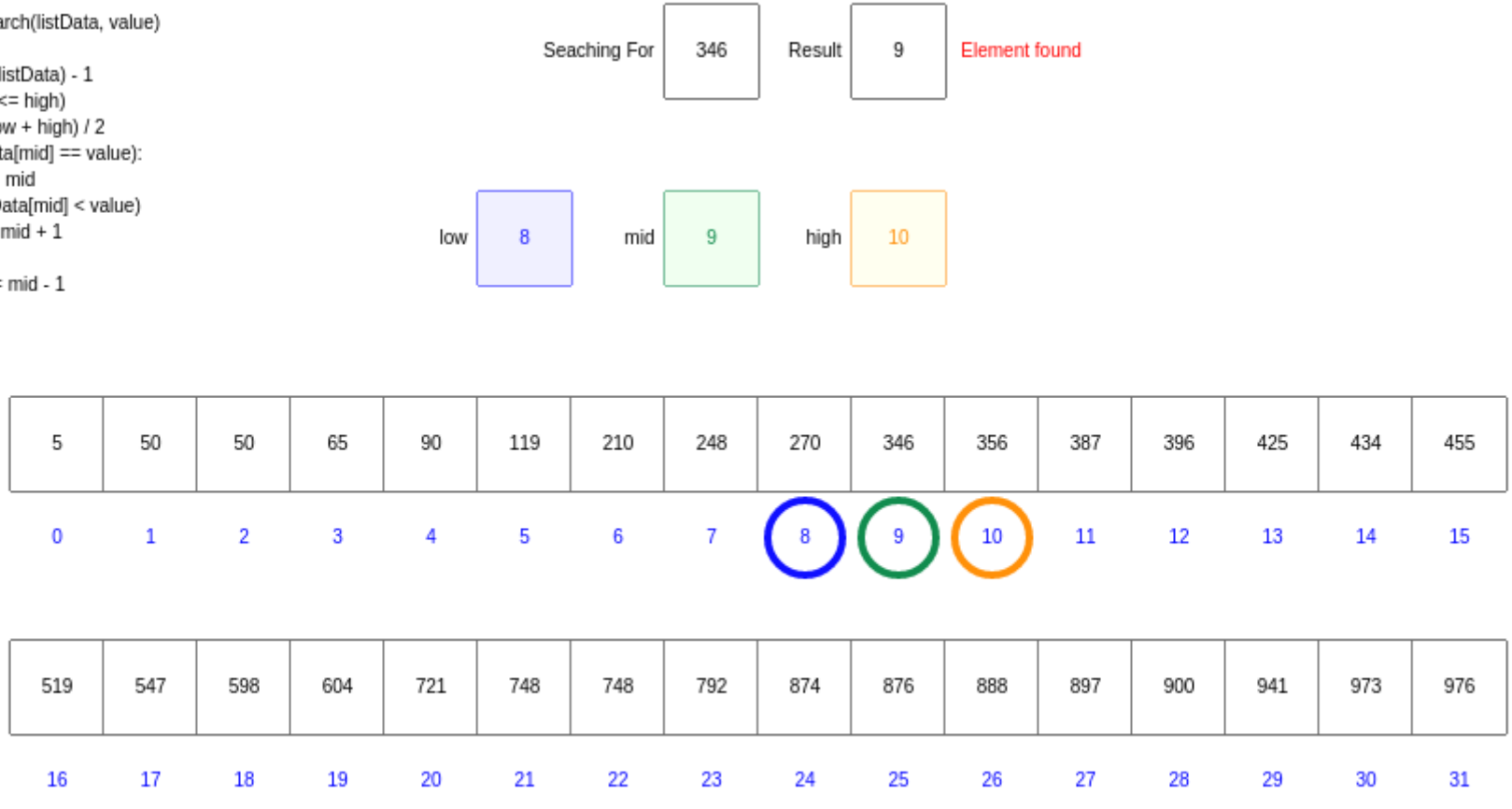
5	50	50	65	90	119	210	248	270	346	356	387	396	425	434	455
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

519	547	598	604	721	748	748	792	874	876	888	897	900	941	973	976
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Linear Search

Binary and Linear Search

```
def binarySearch(listData, value)
    low = 0
    high = len(listData) - 1
    while (low <= high)
        mid = (low + high) / 2
        if (listData[mid] == value):
            return mid
        elif (listData[mid] < value)
            low = mid + 1
        else:
            high = mid - 1
    return -1
```

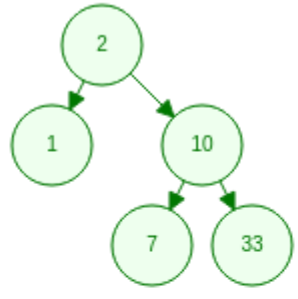


Binary Search

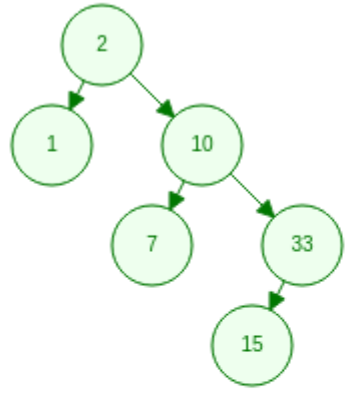
Trees

- Binary Search Trees
- AVL Trees
- Red-Black Trees
- Splay Trees
- Trie (Prefix Tree)
- Radix Tree (Compact Trie)
- Ternary Search Tree (Trie with BST)
- B Trees
- B+ Trees

Binary Search Trees

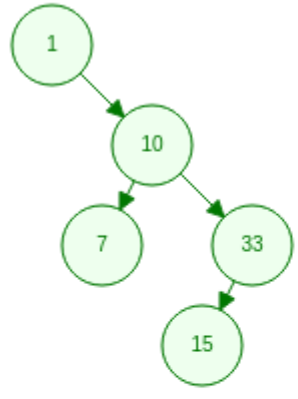


Binary Search Trees



Insert

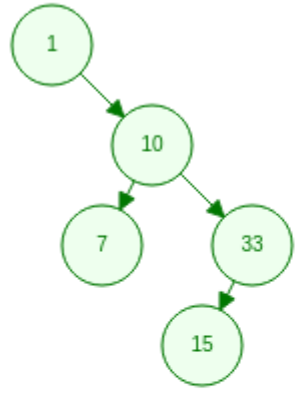
Binary Search Trees



Delete

Binary Search Trees

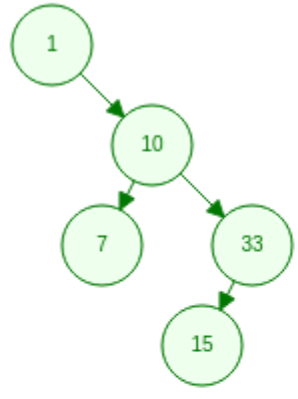
Found:15



Find

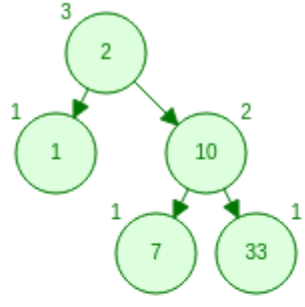
Binary Search Trees

Found:15

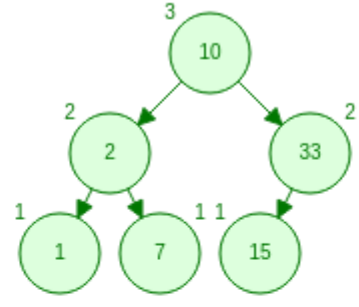


Print

AVL Trees

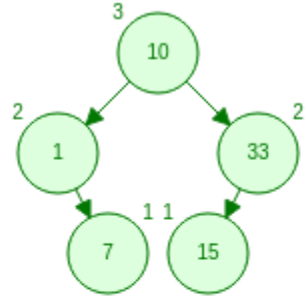


AVL Trees



Insert

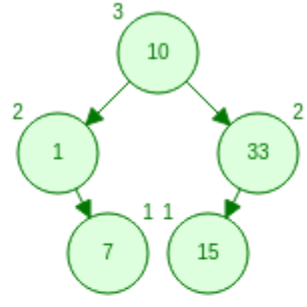
AVL Trees



Delete

AVL Trees

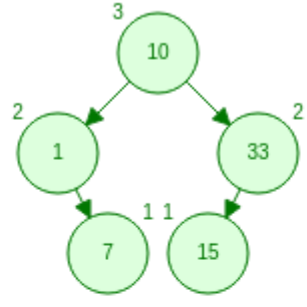
Found:15



Find

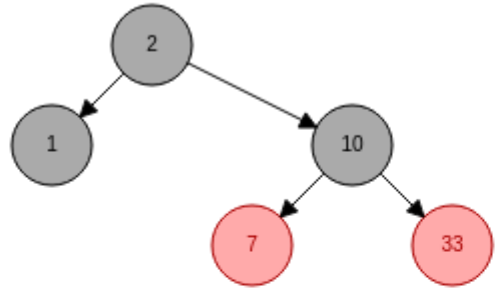
AVL Trees

Found:15

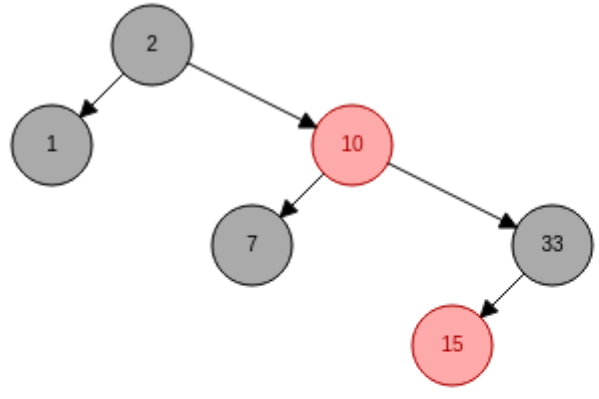


Print

Red-Black Trees

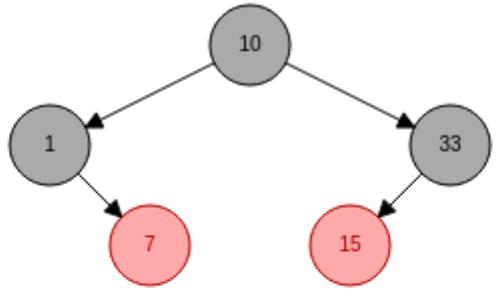


Red-Black Trees



Insert

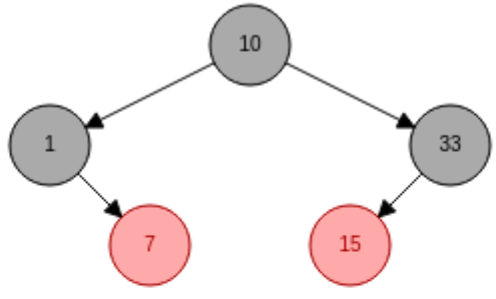
Red-Black Trees



Delete

Red-Black Trees

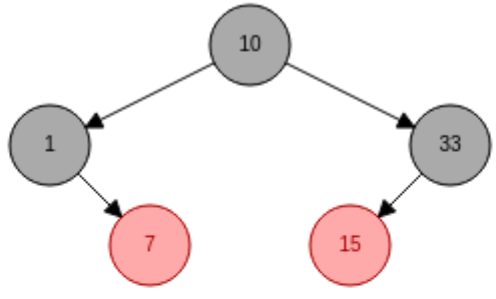
Found:15



Find

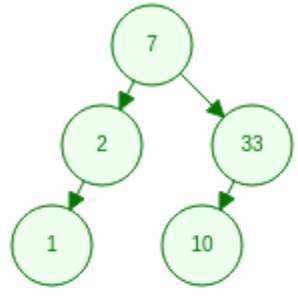
Red-Black Trees

Found:15

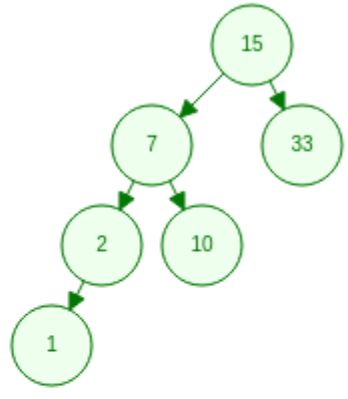


Print

Splay Trees

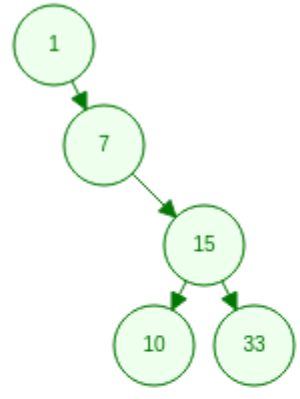


Splay Trees



Insert

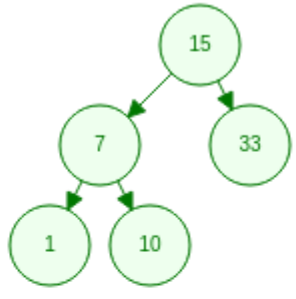
Splay Trees



Delete

Splay Trees

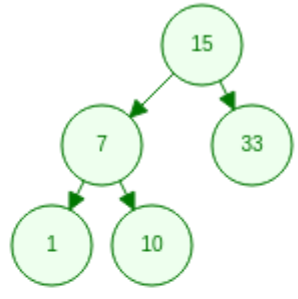
Element 15 found.



Find

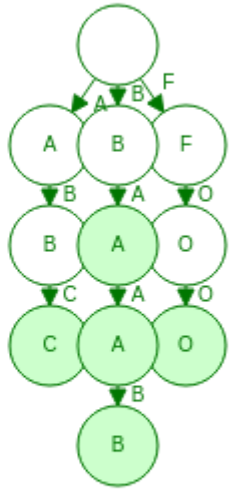
Splay Trees

Element 15 found.

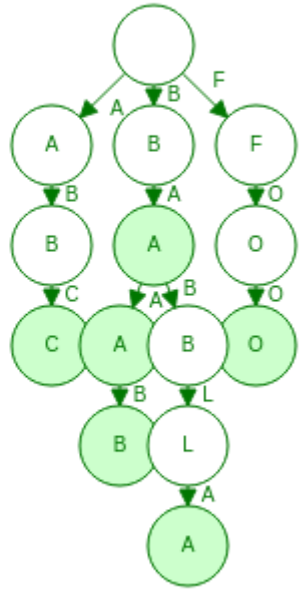


Print

Trie (Prefix Tree)

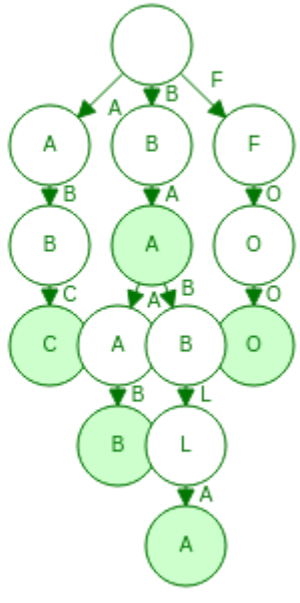


Trie (Prefix Tree)



Insert

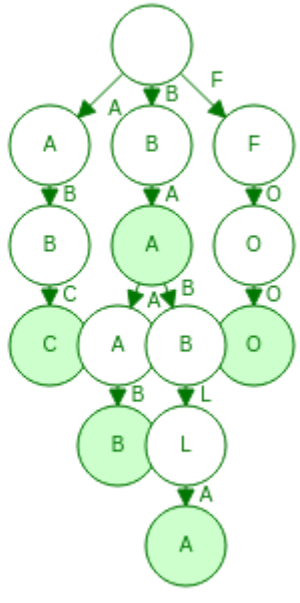
Trie (Prefix Tree)



Delete

Trie (Prefix Tree)

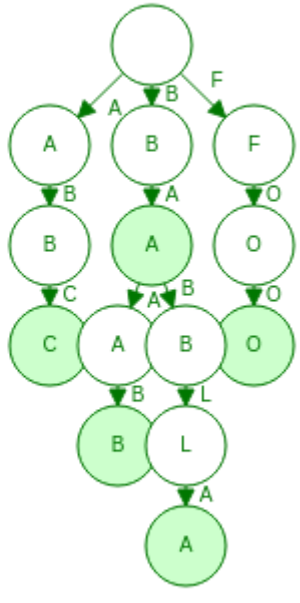
Found "BABLA"



Find

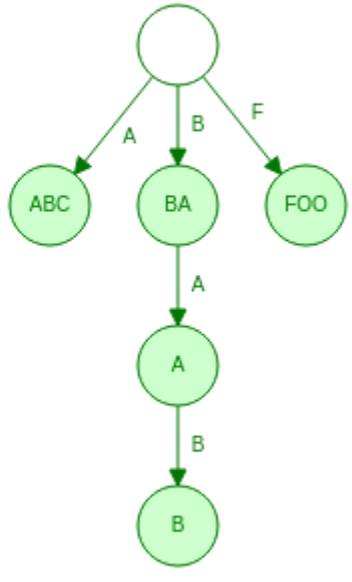
Trie (Prefix Tree)

Found "BABLA"

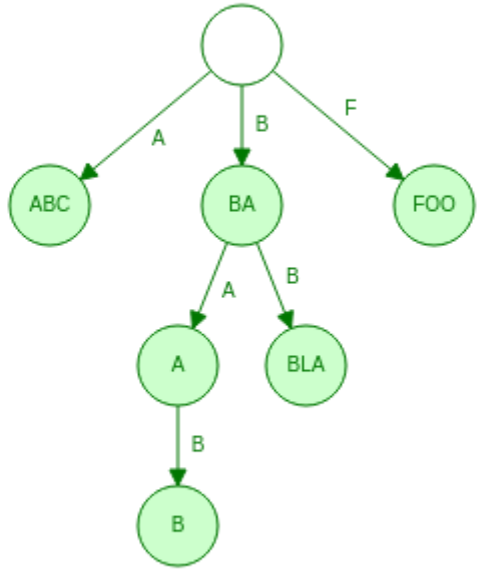


Print

Radix Tree (Compact Trie)

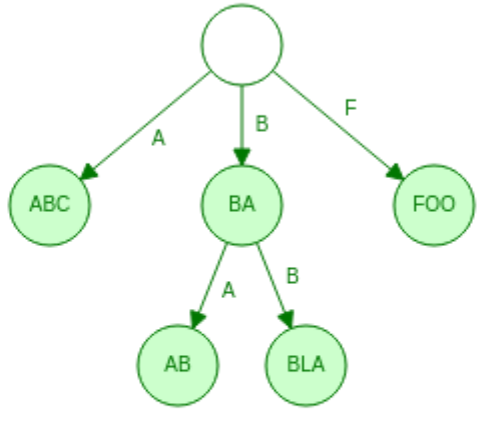


Radix Tree (Compact Trie)



Insert

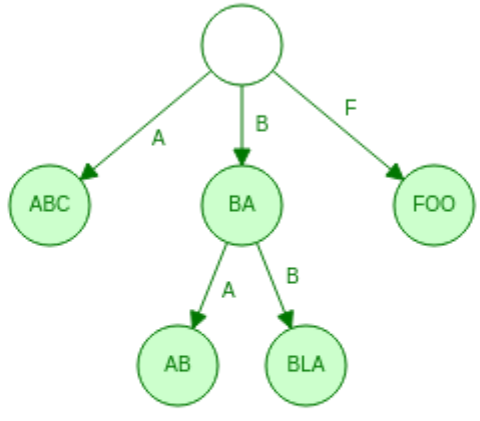
Radix Tree (Compact Trie)



Delete

Radix Tree (Compact Trie)

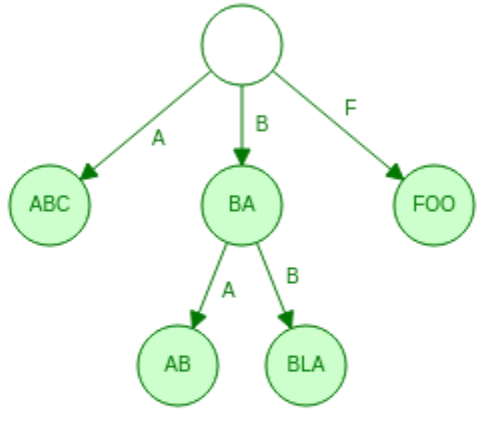
String BABLA found



Find

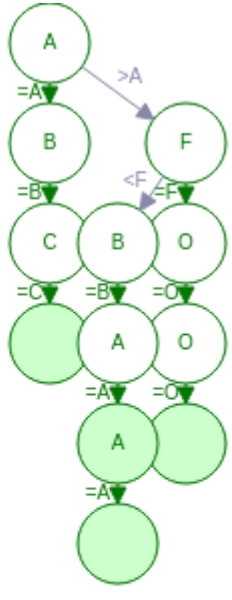
Radix Tree (Compact Trie)

String BABLA found

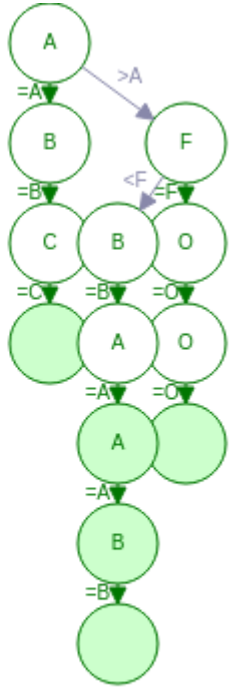


Print

Ternary Search Tree

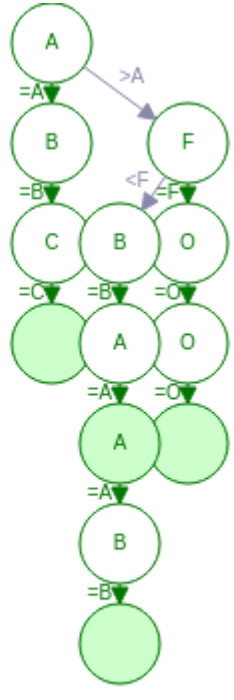


Ternary Search Tree



Insert

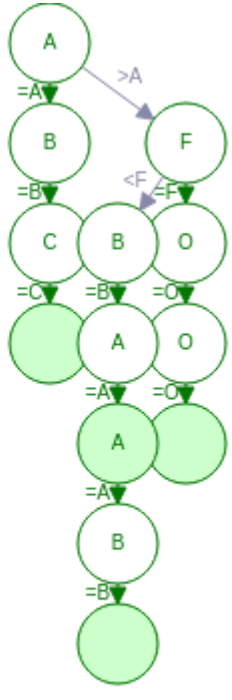
Ternary Search Tree



Delete

Ternary Search Tree

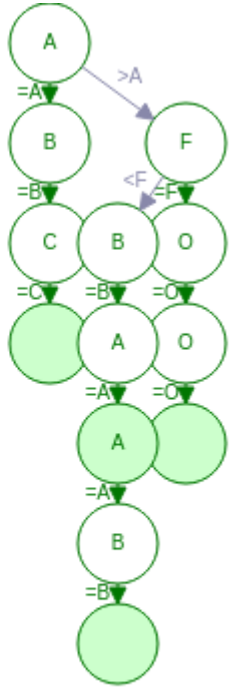
Found "BAAB"



Find

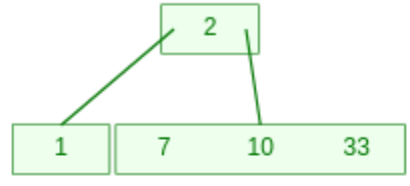
Ternary Search Tree

Found "BAAB"

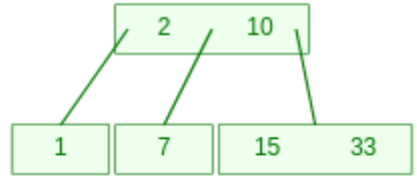


Print

B Trees

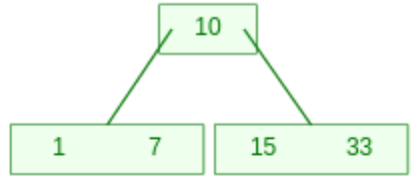


B Trees



Insert

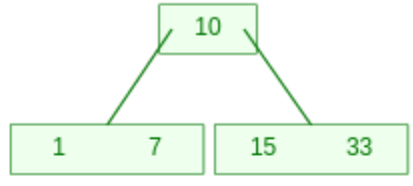
B Trees



Delete

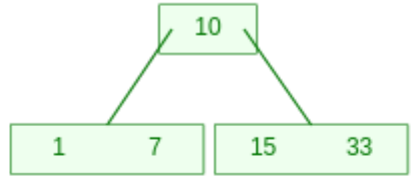
B Trees

Element 15 found



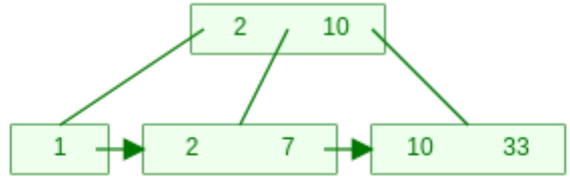
Find

B Trees

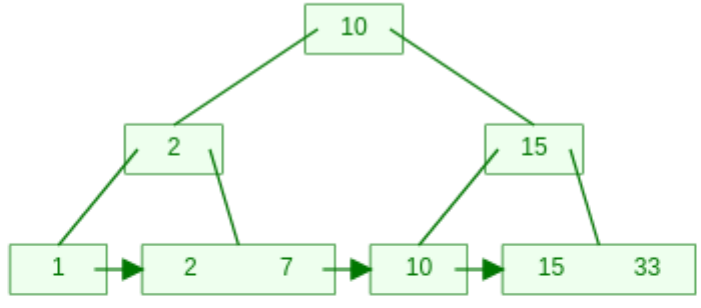


Print

B+ Trees

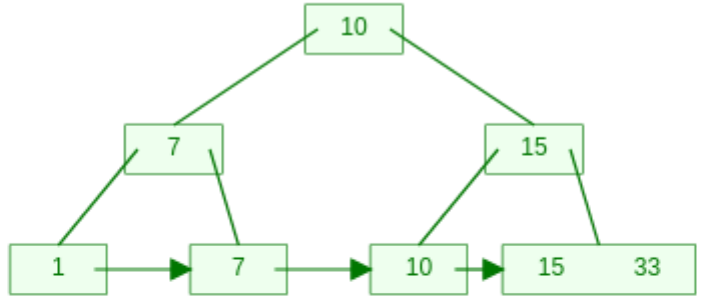


B+ Trees



Insert

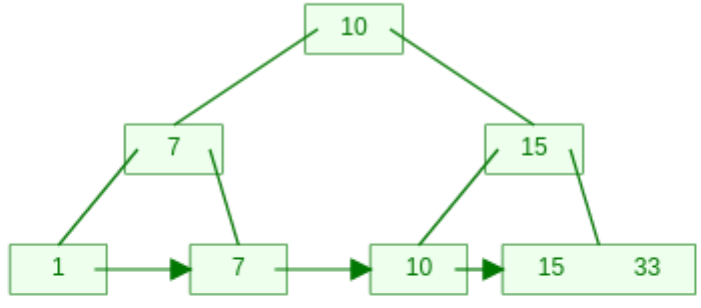
B+ Trees



Delete

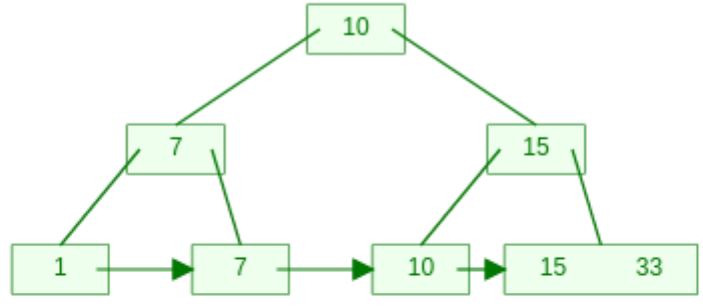
B+ Trees

Element 15 found



Find

B+ Trees

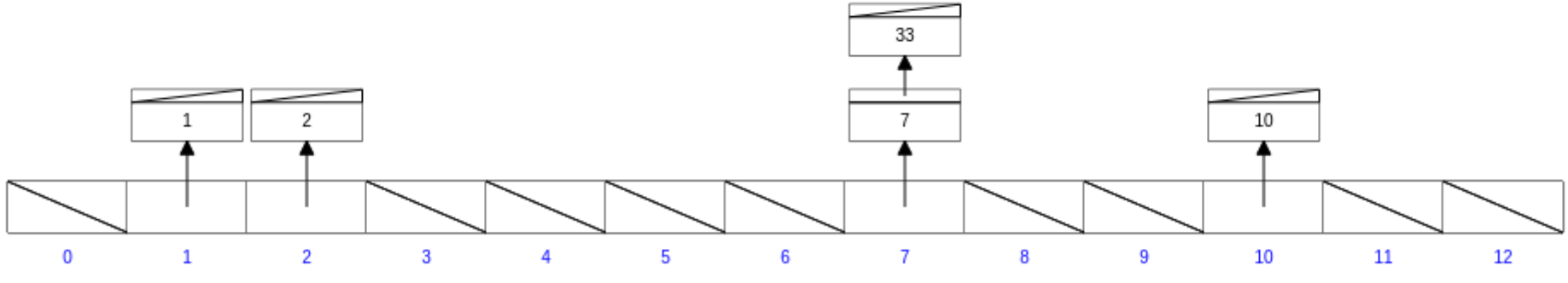


Print

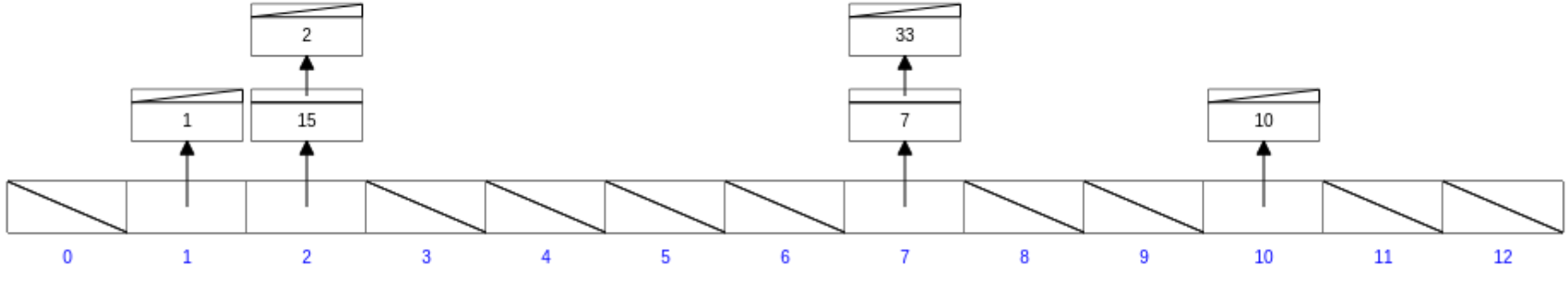
Hashing

- Open Hash Tables (Closed Addressing)
- Closed Hash Tables (Open Addressing)
- Closed Hash Tables, using buckets

Open Hash Tables (Closed Addressing)



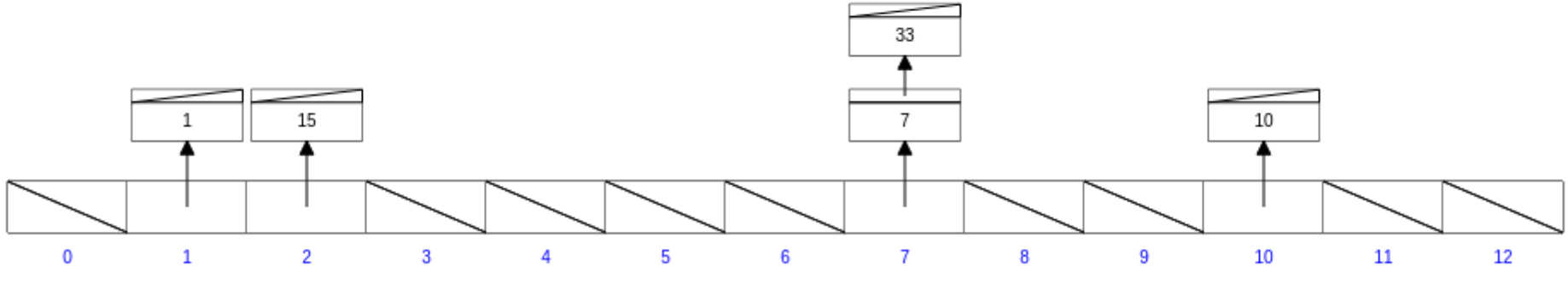
Open Hash Tables (Closed Addressing)



Insert

Open Hash Tables (Closed Addressing)

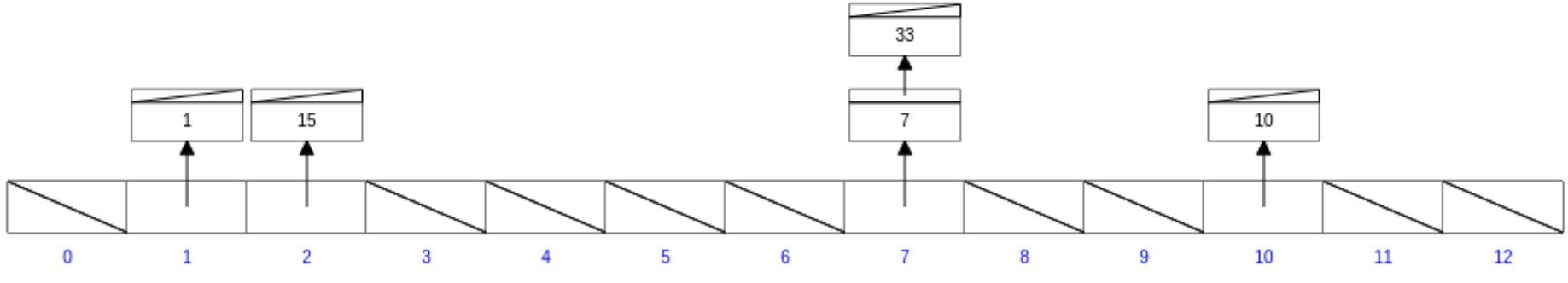
Deleting element: 2 Element deleted



Delete

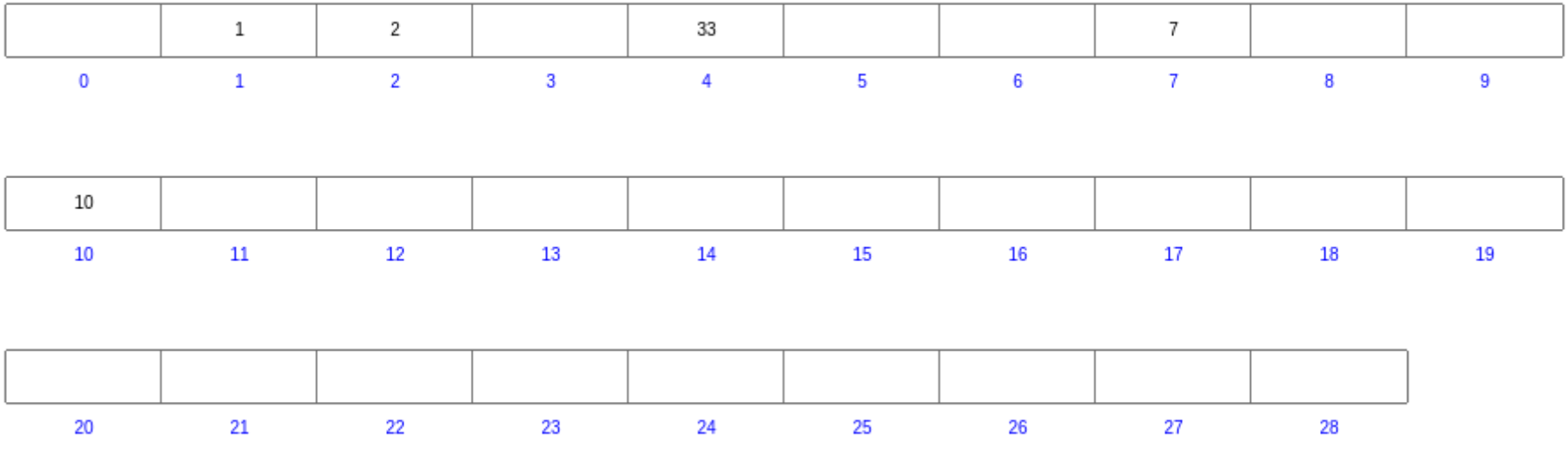
Open Hash Tables (Closed Addressing)

Finding Element: 15 Found!

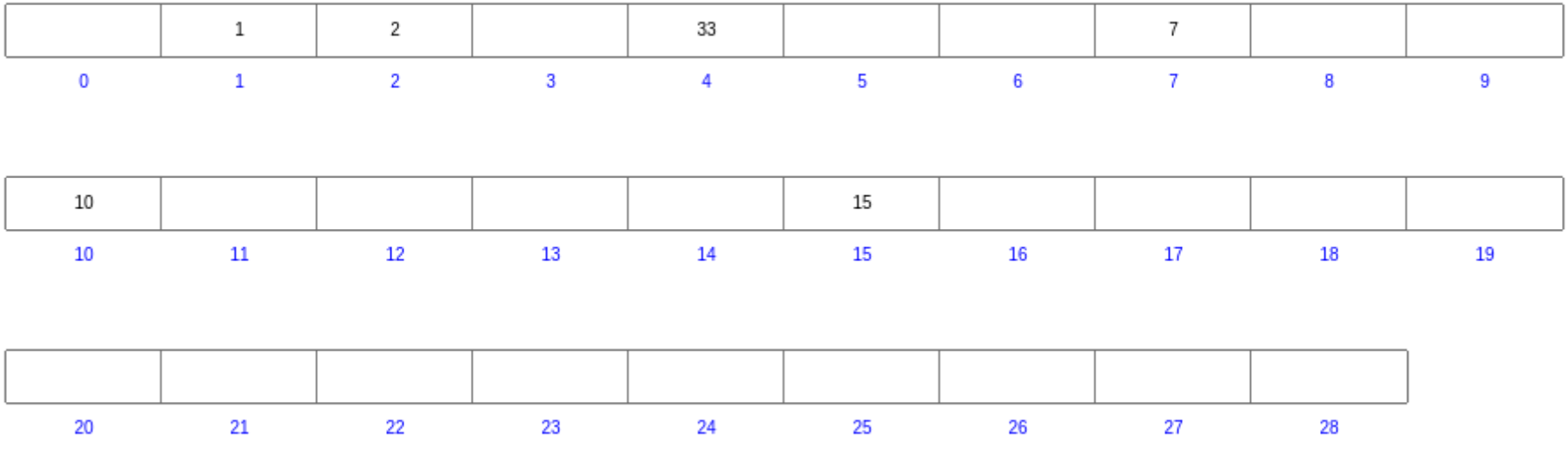


Find

Closed Hash Tables (Open Addressing)



Closed Hash Tables (Open Addressing)



Insert

Closed Hash Tables (Open Addressing)

Deleting element: 2 Element deleted

	1	<deleted>		33			7		
0	1	2	3	4	5	6	7	8	9

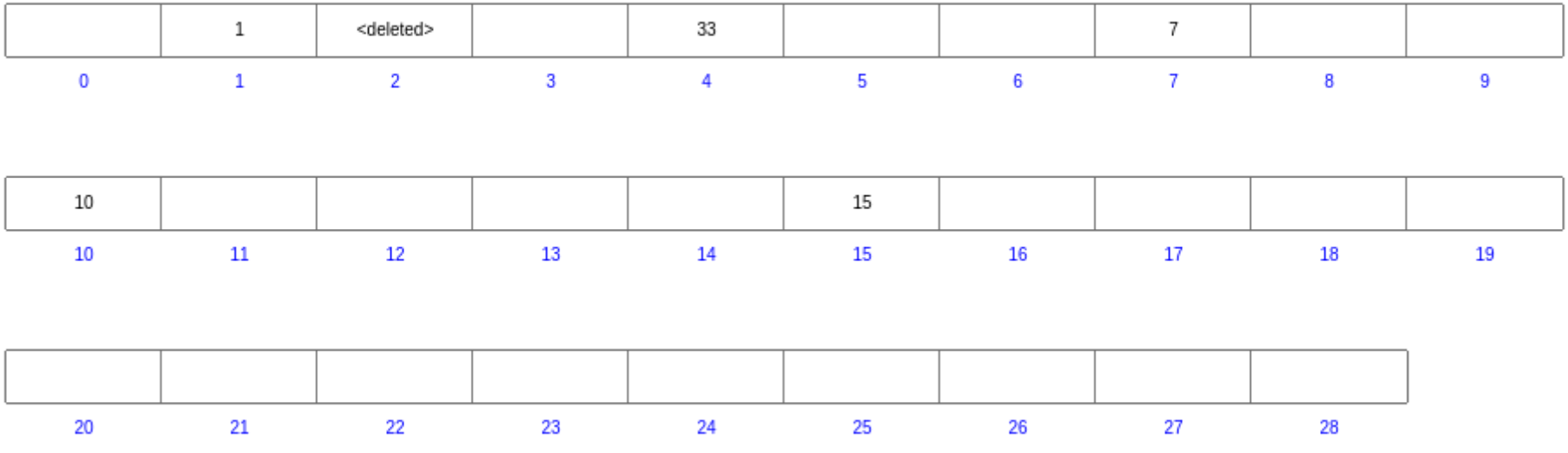
10					15				
10	11	12	13	14	15	16	17	18	19

20	21	22	23	24	25	26	27	28

Delete

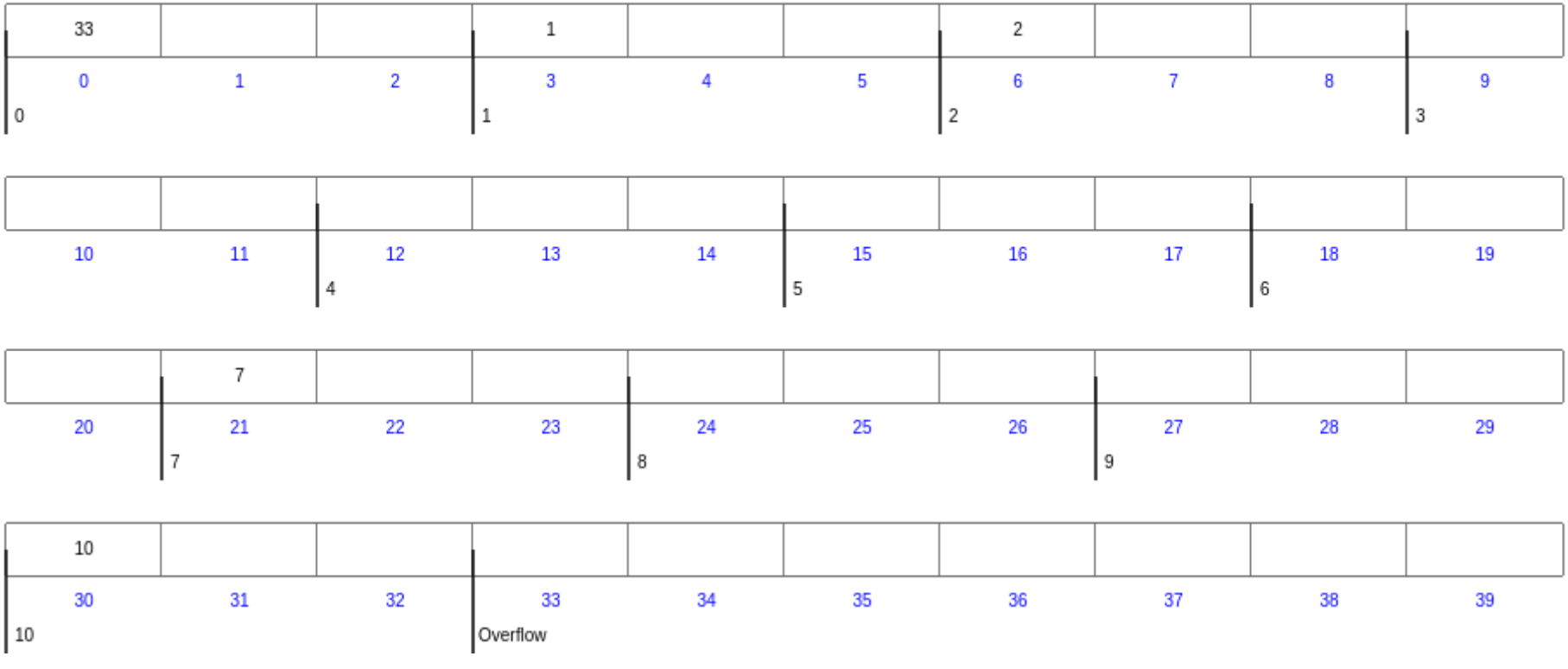
Closed Hash Tables (Open Addressing)

Finding Element: 15 Found!

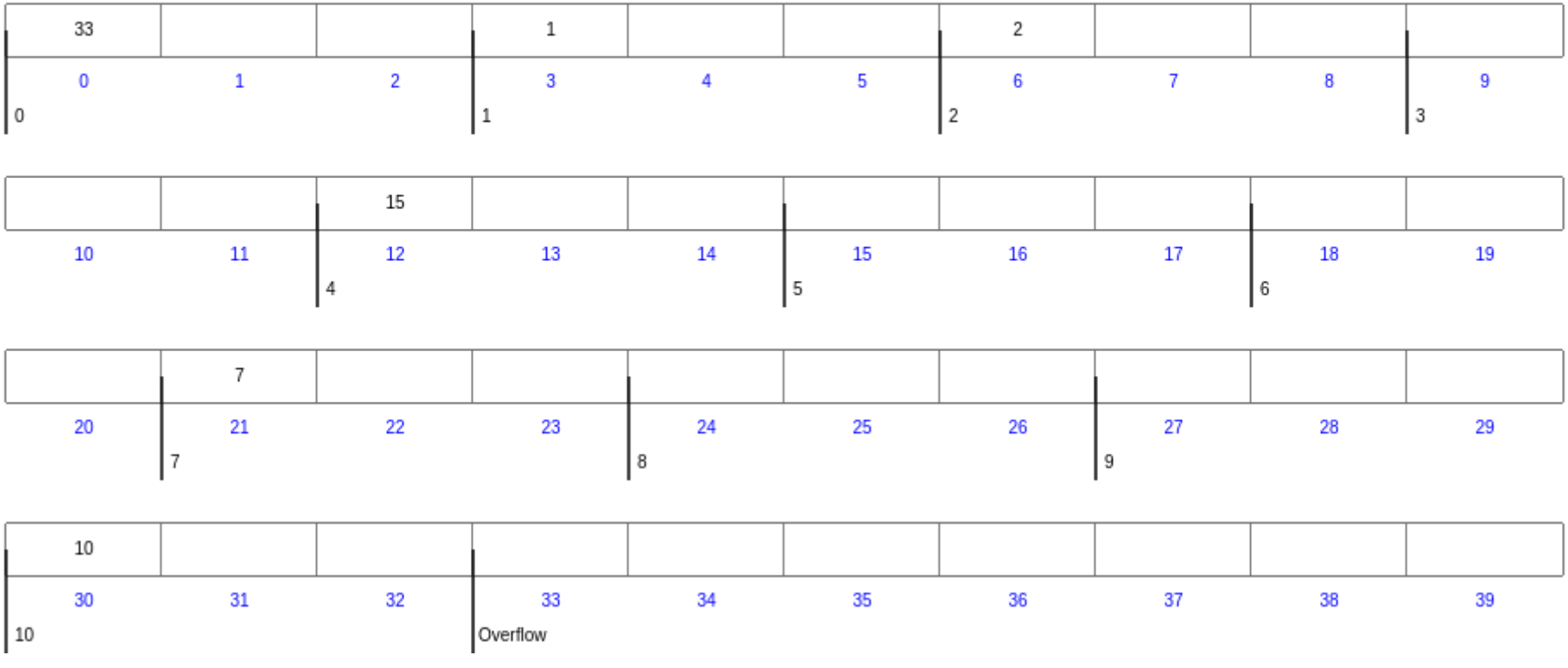


Find

Closed Hash Tables, using buckets

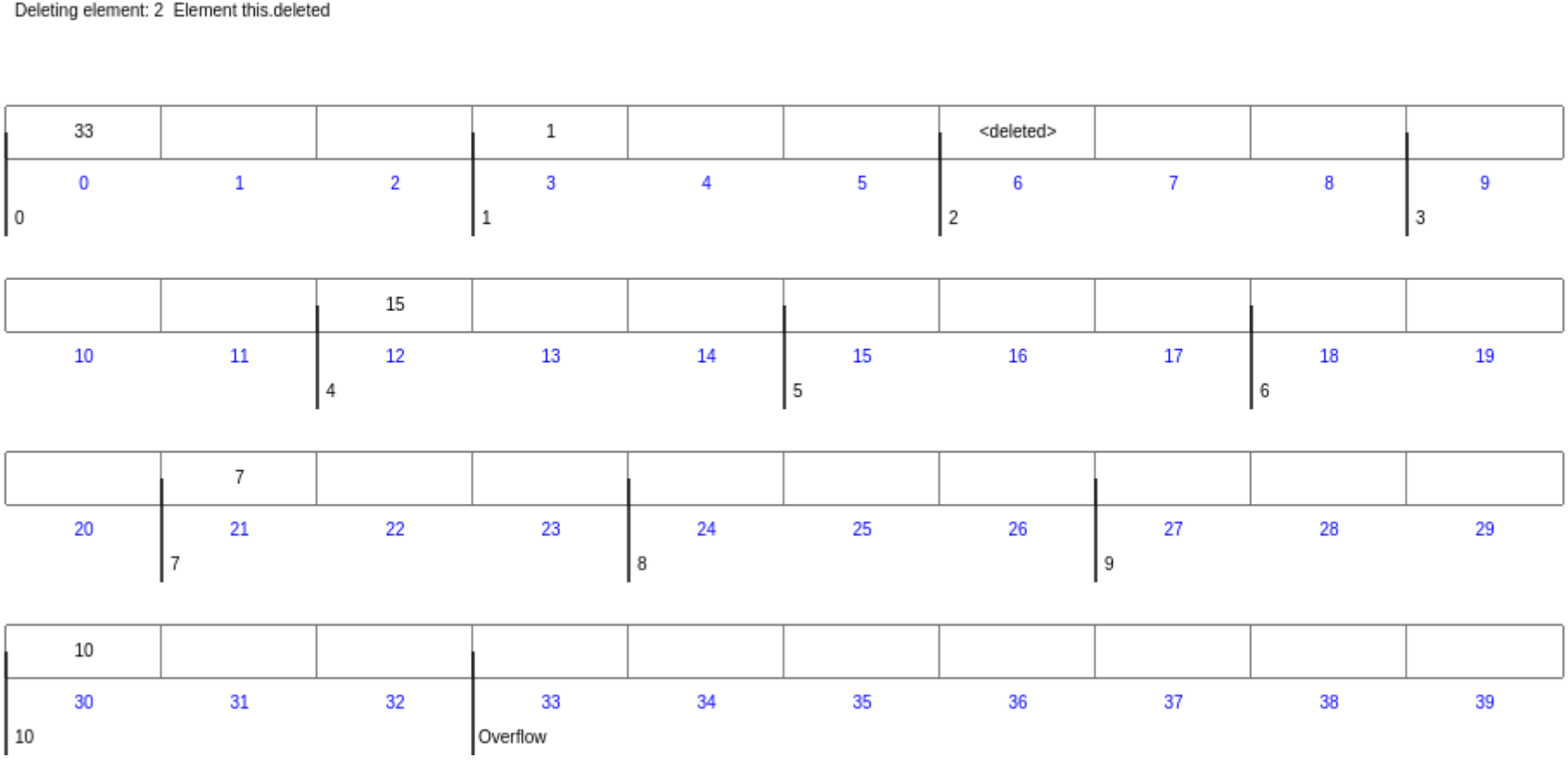


Closed Hash Tables, using buckets



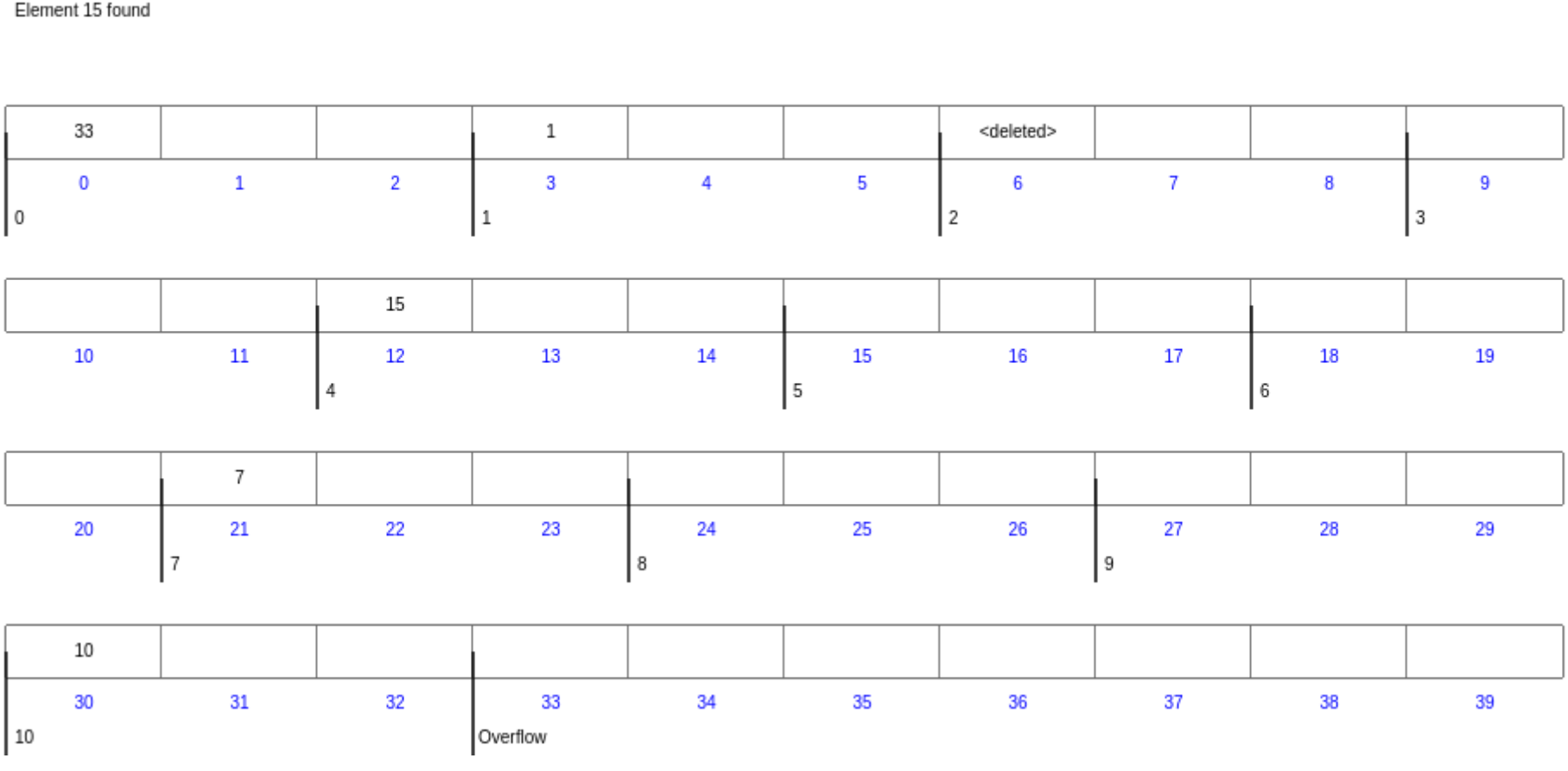
Insert

Closed Hash Tables, using buckets



Delete

Closed Hash Tables, using buckets



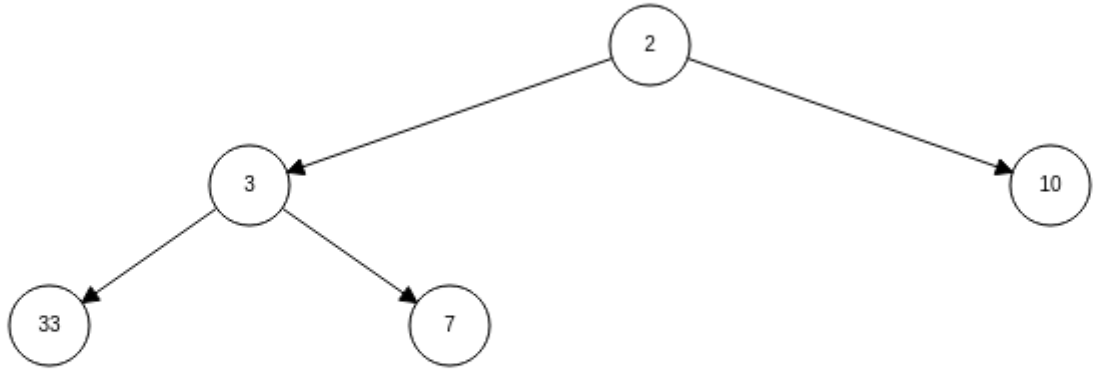
Find

Heaps

- Heaps
- Binomial Queues
- Fibonacci Heaps
- Leftist Heaps
- Skew Heaps

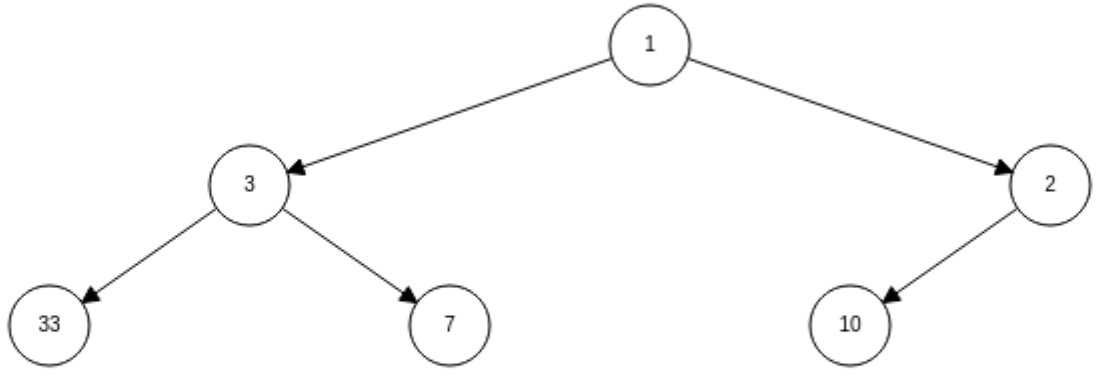
Heaps

-INF	2	3	10	33	7																										
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31



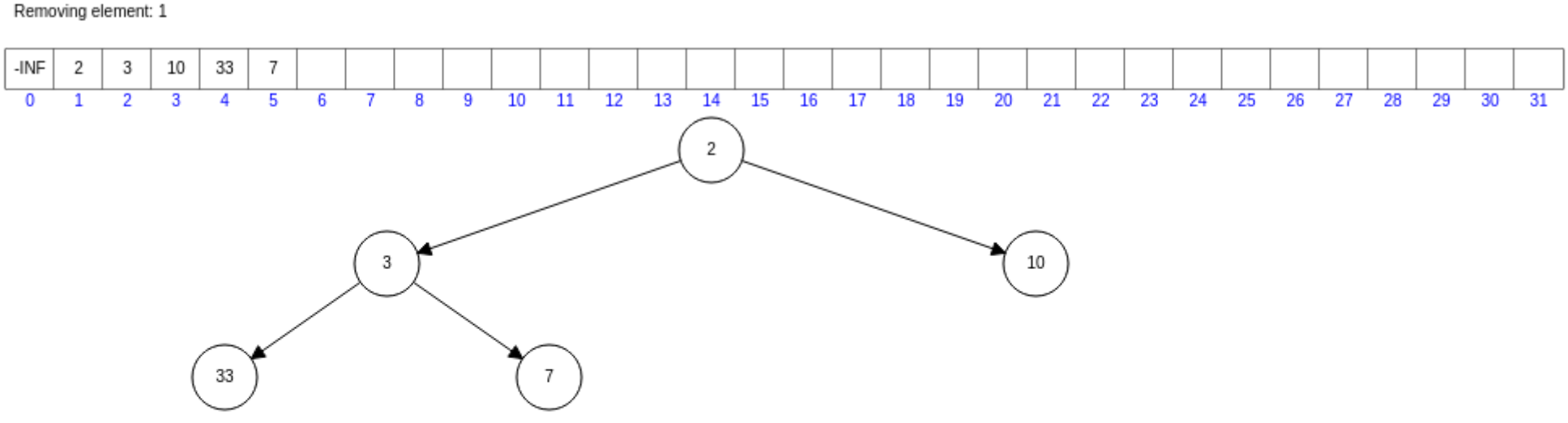
Heaps

-INF	1	3	2	33	7	10																							
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31



Insert

Heaps

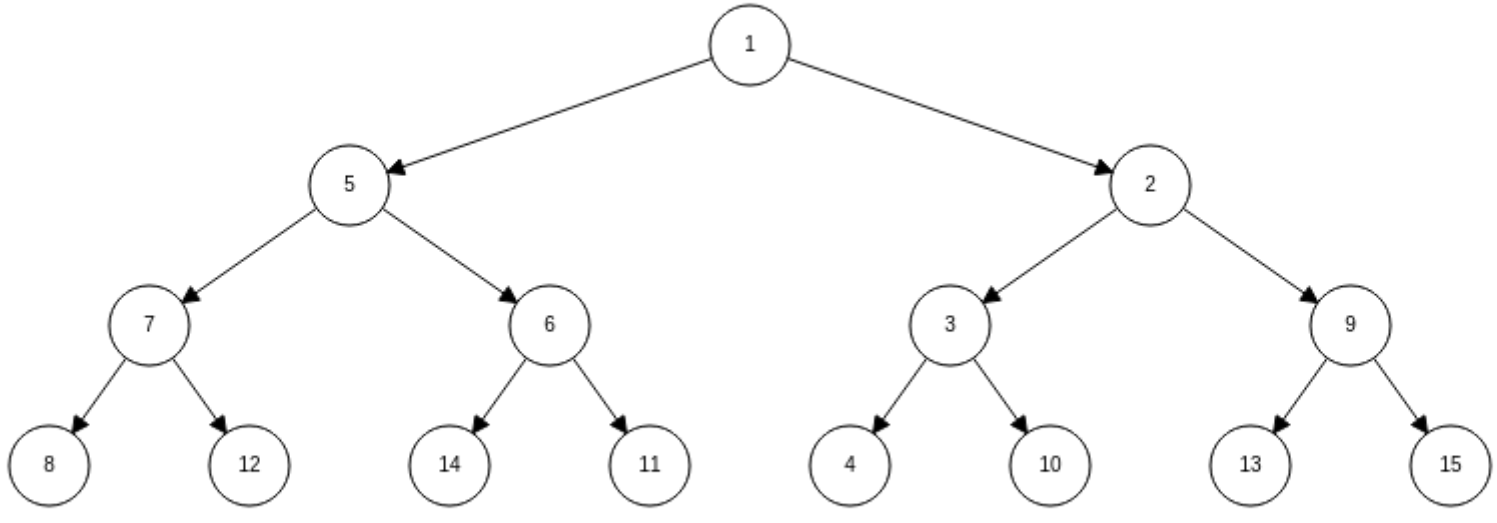


Remove smallest

Heaps

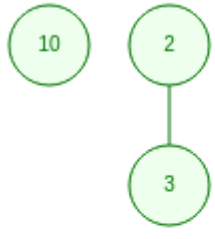
Removing element: 1

-INF	1	5	2	7	6	3	9	8	12	14	11	4	10	13	15																
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

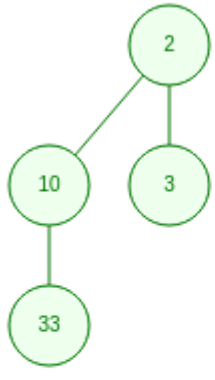


Build heap

Binomial Queues

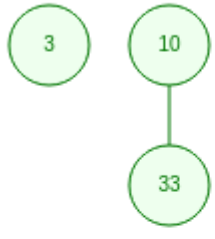


Binomial Queues



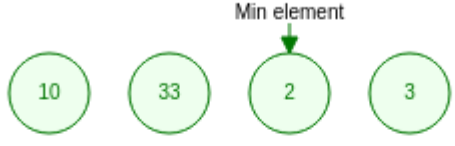
Insert

Binomial Queues



Remove smallest

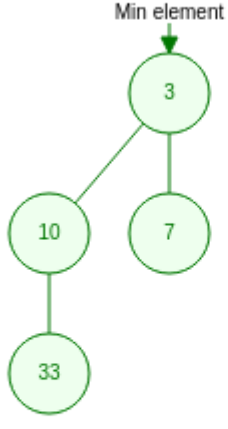
Fibonacci Heaps



Fibonacci Heaps

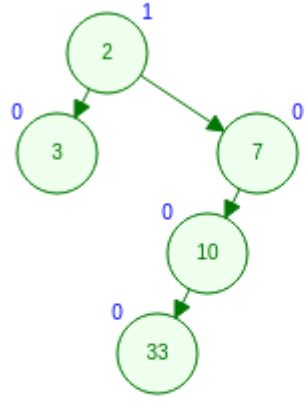
Insert

Fibonacci Heaps

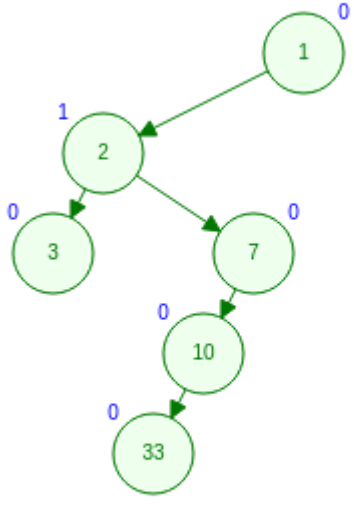


Remove smallest

Leftist Heaps

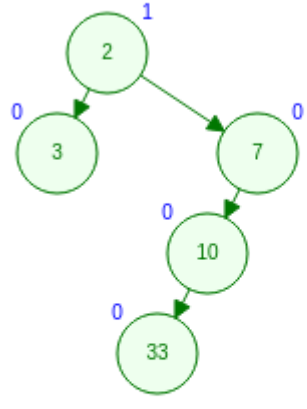


Leftist Heaps



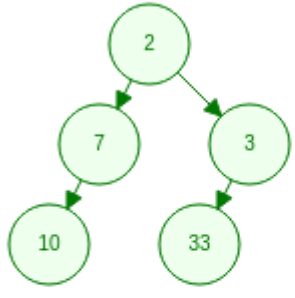
Insert

Leftist Heaps

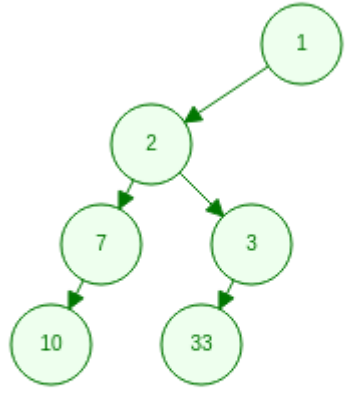


Remove smallest

Skew Heaps

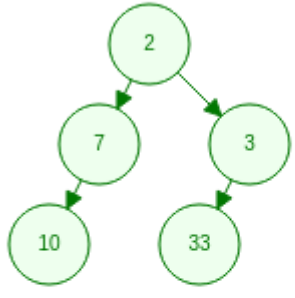


Skew Heaps



Insert

Skew Heaps



Remove smallest

Graph Algorithms

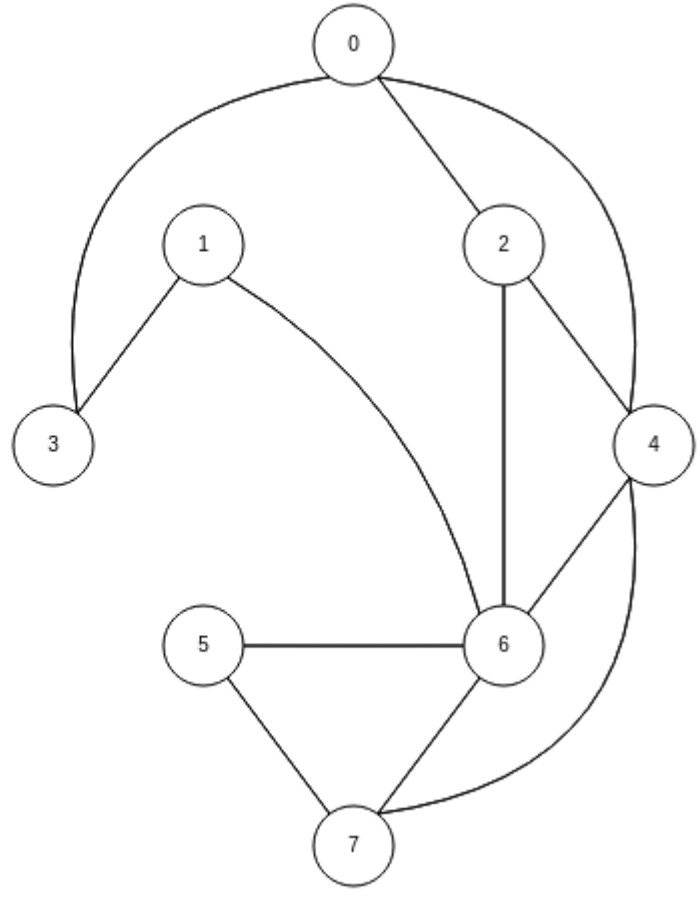
- Breadth-First Search
- Depth-First Search
- Connected Components
- Dijkstra's Shortest Path
- Prim's Minimum Cost Spanning Tree
- Topological Sort (using Indegree array)
- Topological Sort (using DFS)
- Floyd-Warshall (all pairs shortest paths)
- Kruskal Minimum Cost Spanning Tree

Breadth-First Search

BFS Queue

	Parent
0	
1	
2	
3	
4	
5	
6	
7	

	Visited
0	f
1	f
2	f
3	f
4	f
5	f
6	f
7	f

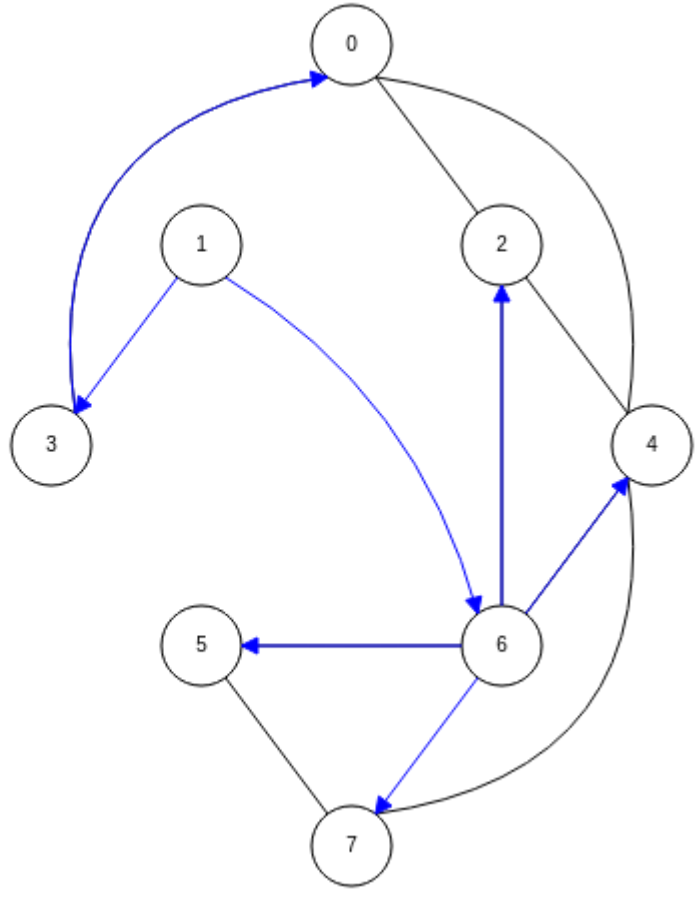


Breadth-First Search

BFS Queue

	Parent
0	3
1	
2	6
3	1
4	6
5	6
6	1
7	6

	Visited
0	T
1	f
2	T
3	T
4	T
5	T
6	T
7	T

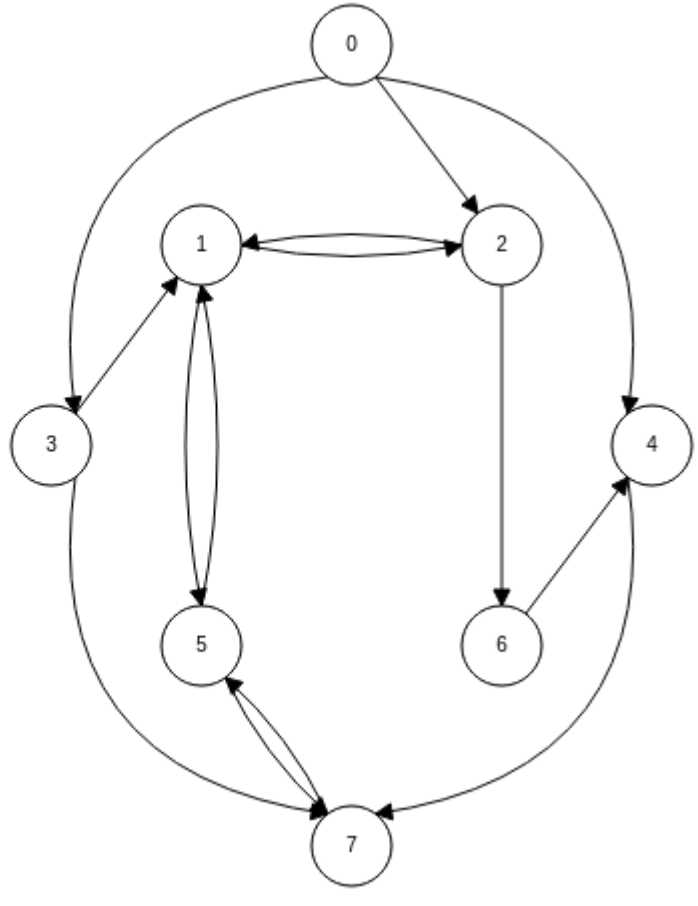


BFS from 1

Depth-First Search

	Parent
0	
1	
2	
3	
4	
5	
6	
7	

	Visited
0	f
1	f
2	f
3	f
4	f
5	f
6	f
7	f

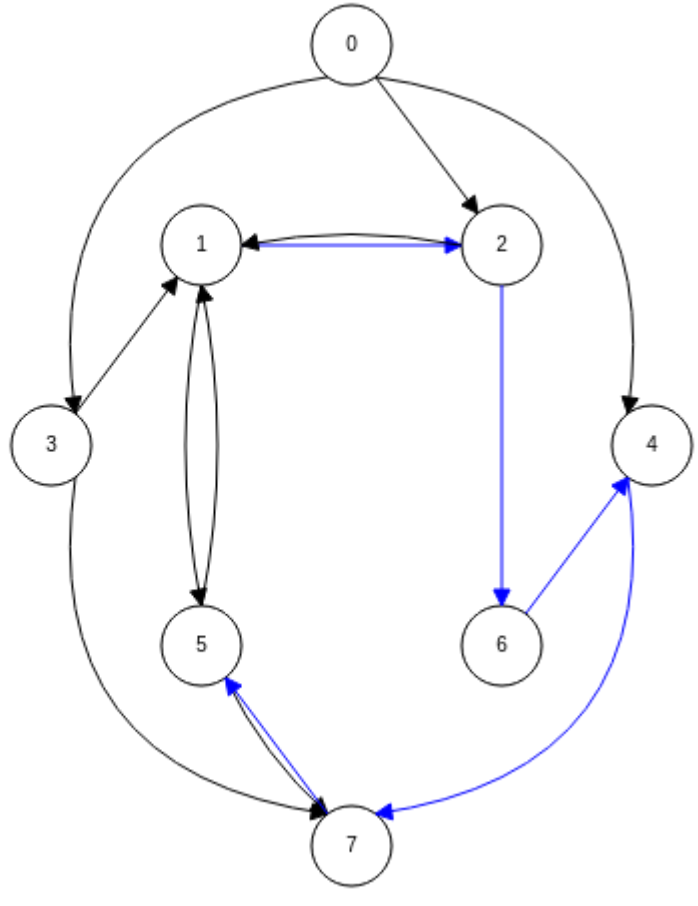


Depth-First Search

DFS(1)
DFS(2)
DFS(6)
DFS(4)
DFS(7)
DFS(5)

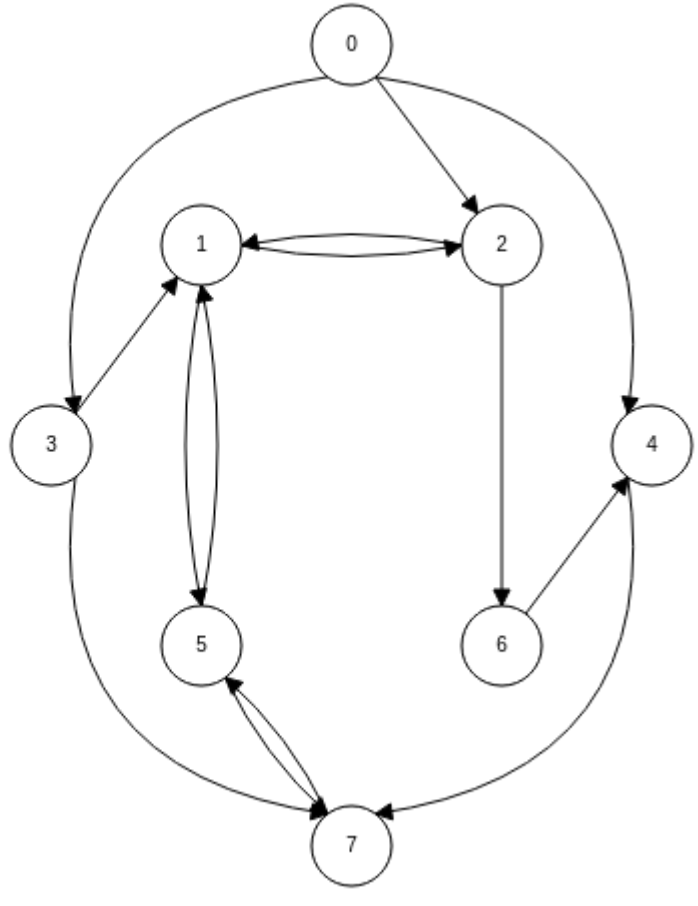
	Parent
0	
1	
2	1
3	
4	6
5	7
6	2
7	4

	Visited
0	f
1	T
2	T
3	f
4	T
5	T
6	T
7	T



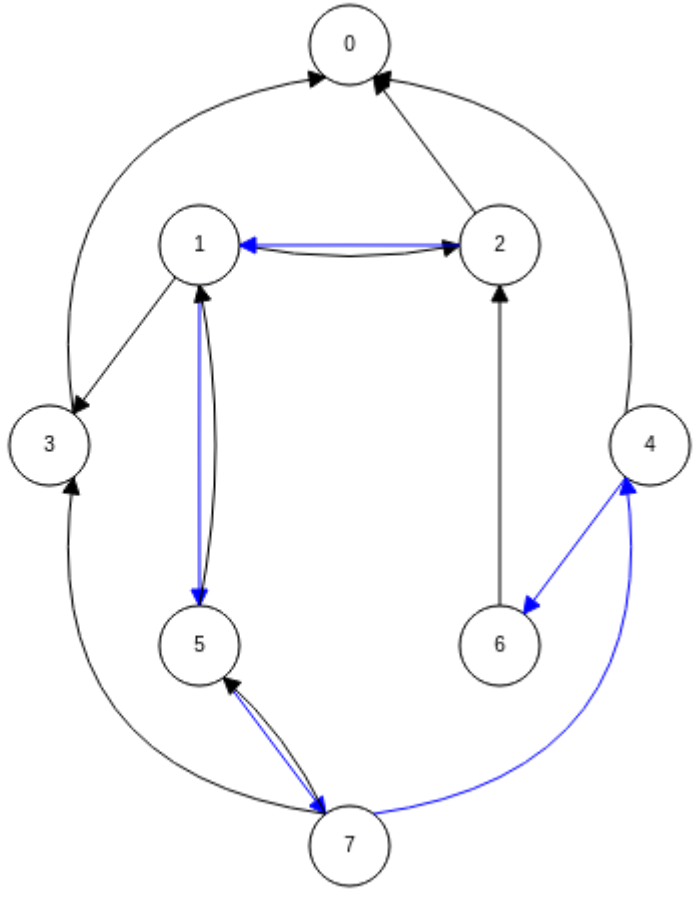
DFS from 1

Connected Components



Connected Components

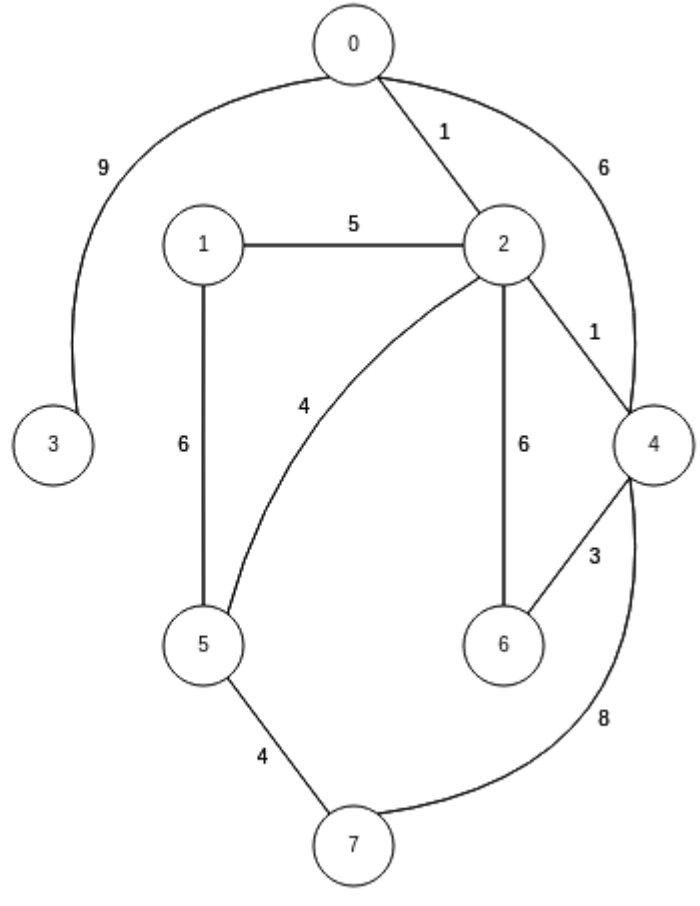
CC #1
Vertex 0 DFS(0)
CC #2
Vertex 3 DFS(3)
CC #3
Vertex 2 DFS(2)
Vertex 1 DFS(1)
Vertex 5 DFS(5)
Vertex 7 DFS(7)
Vertex 4 DFS(4)
Vertex 6 DFS(6)



Connected components

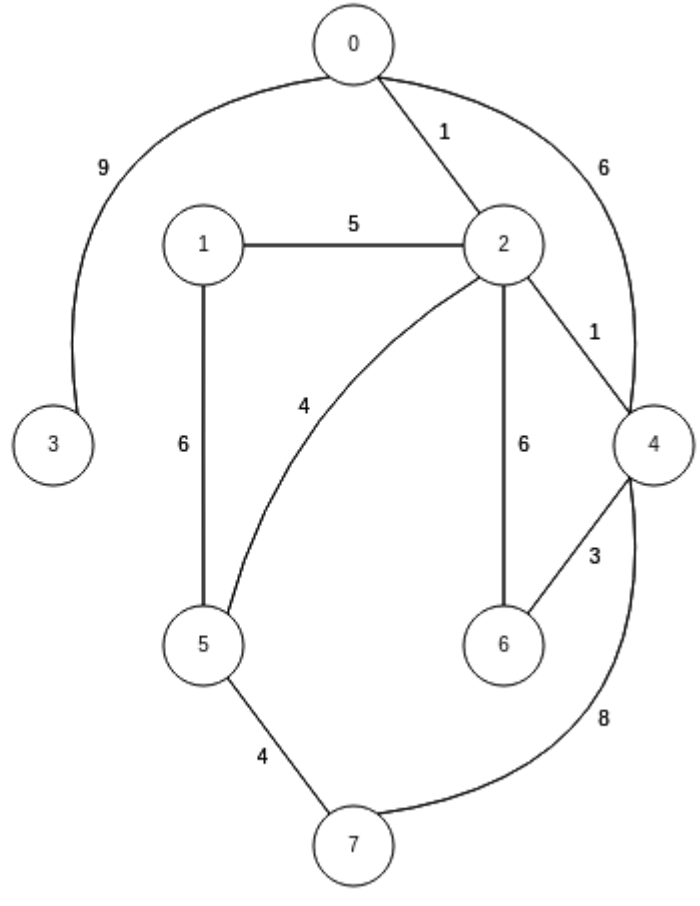
Dijkstra's Shortest Path

Vertex	Known	Cost	Path



Dijkstra's Shortest Path

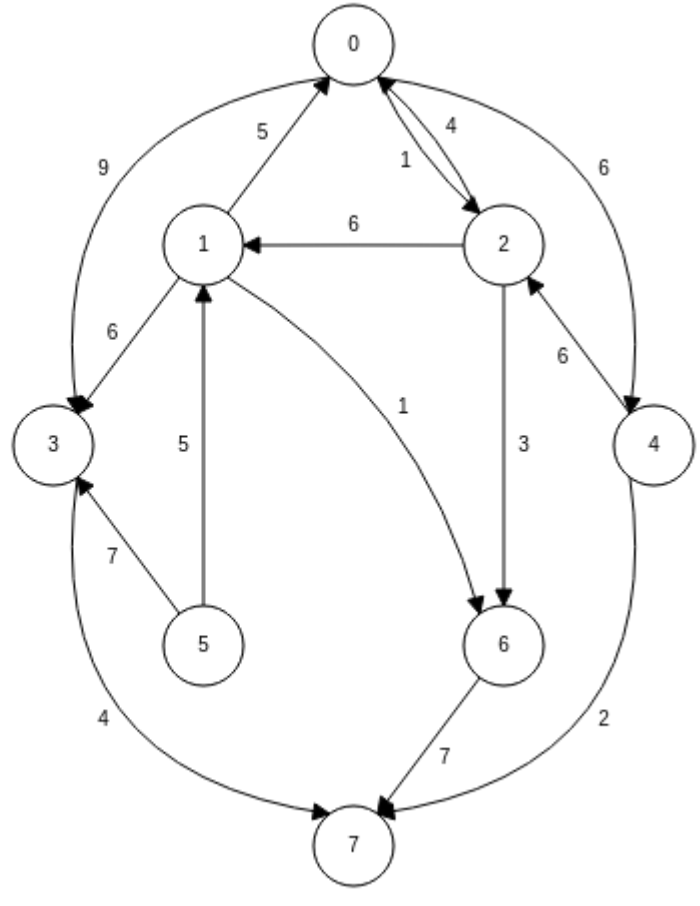
Vertex	Known	Cost	Path	
	T	6	2	1 2 0
	T	0	-1	1
	T	5	1	1 2
	T	15	0	1 2 0 3
	T	6	2	1 2 4
	T	6	1	1 5
	T	9	4	1 2 4 6
	T	10	5	1 5 7



Dijkstra from 1

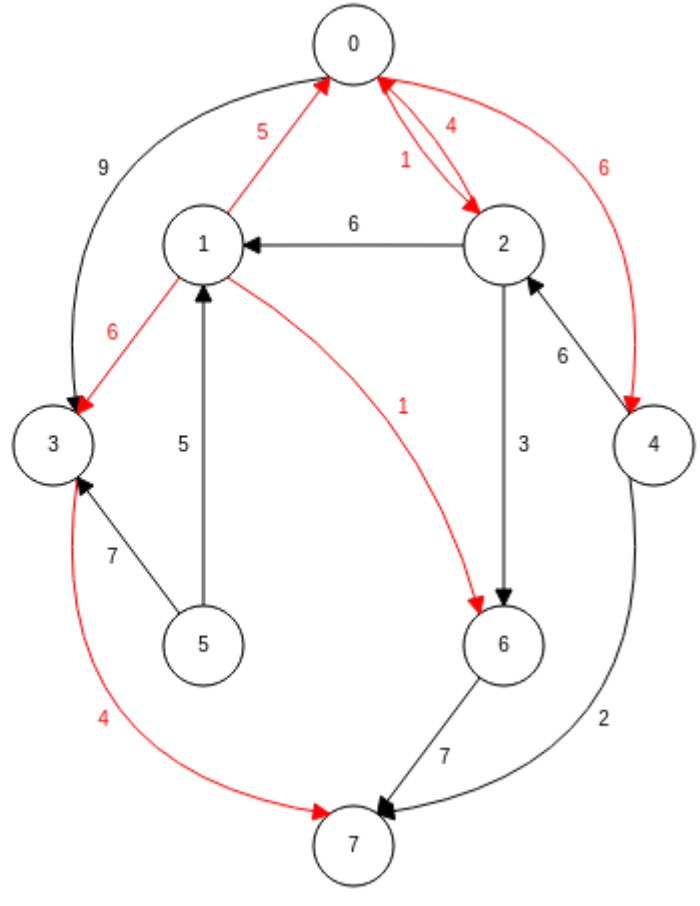
Prim's Minimum Cost Spanning Tree

Vertex	Known	Cost	Path



Prim's Minimum Cost Spanning Tree

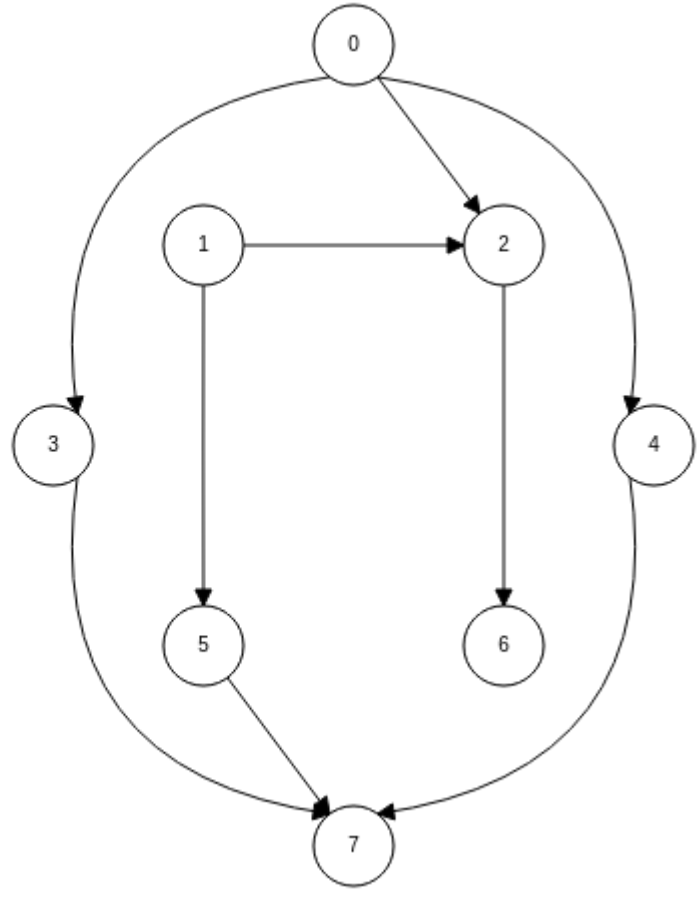
Vertex	Known	Cost	Path
	T	5	1
	T	0	-1
	T	1	0
	T	6	1
	T	6	0
	F	INF	-1
	T	1	1
	T	4	3



Prim from 1

Topological Sort (Indegree)

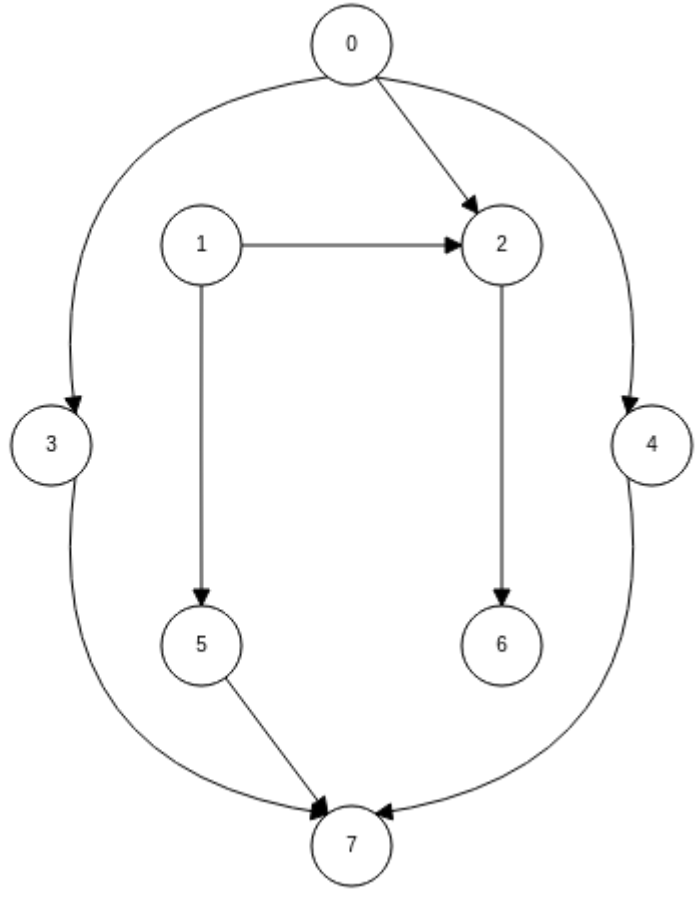
Indegree	
0	
1	
2	
3	
4	
5	
6	
7	



Topological Sort (Indegree)

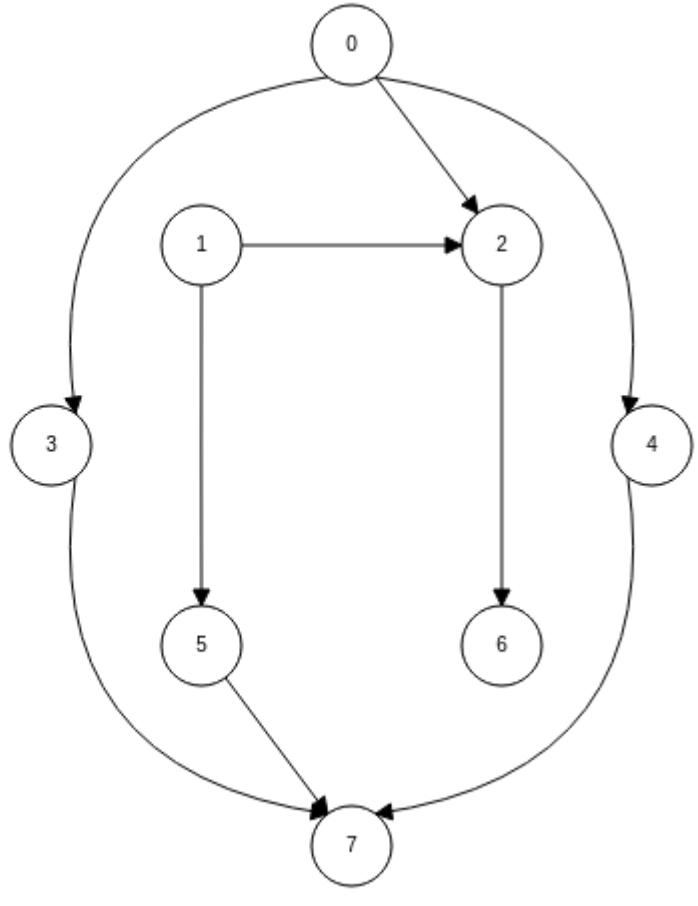
	Indegree
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0

Topological Order
1
5
0
4
3
7
2
6



Topological Sort

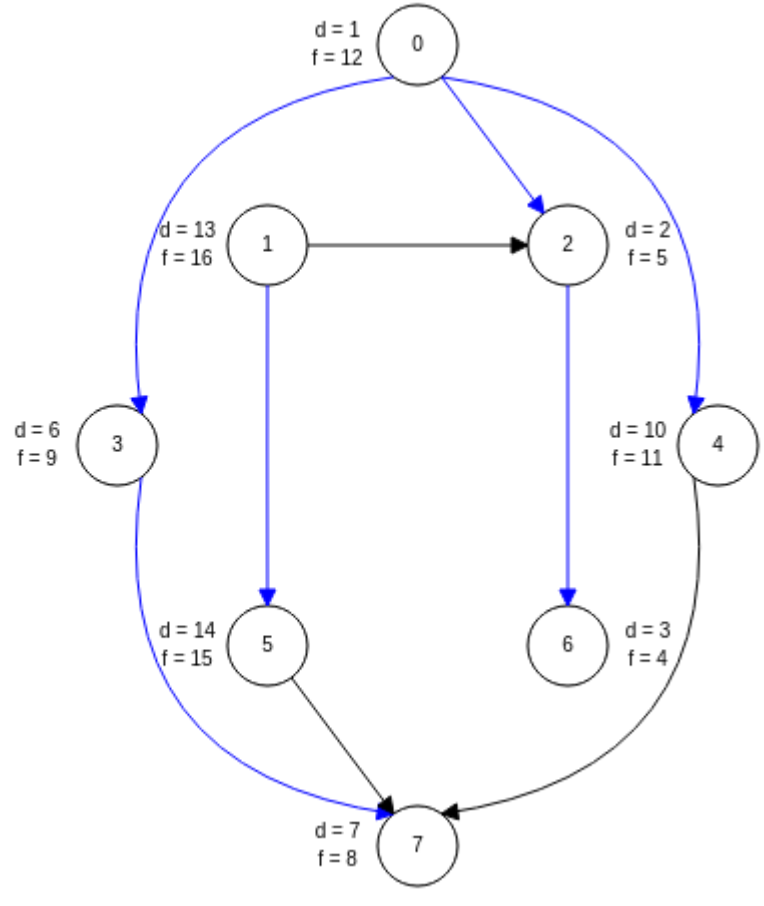
Topological Sort (DFS)



Topological Sort (DFS)

```
DFS(0)
  DFS(2)
    DFS(6)
  DFS(3)
    DFS(7)
  DFS(4)
---
DFS(1)
  DFS(5)
```

Topological Order
1
5
0
4
3
7
2
6



Topological Sort

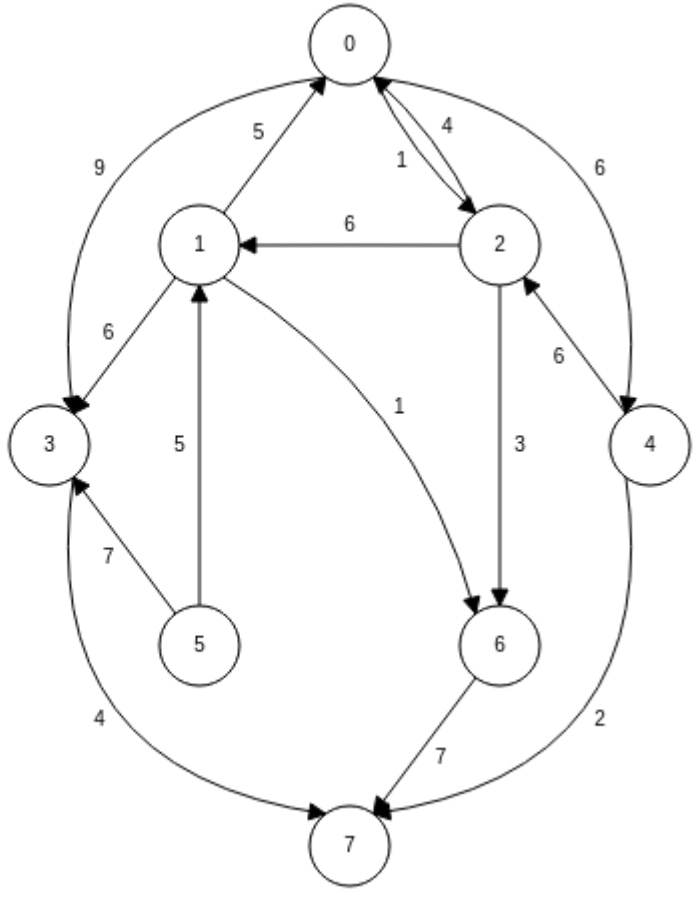
Floyd-Warshall All-Pairs Shortest Path

Cost Table

	0	1	2	3	4	5	6	7
0	INF	INF	1	9	6	INF	INF	INF
1	5	INF	INF	6	INF	INF	1	INF
2	4	6	INF	INF	INF	INF	3	INF
3	INF	INF	INF	INF	INF	INF	INF	4
4	INF	INF	6	INF	INF	INF	INF	2
5	INF	5	INF	7	INF	INF	INF	INF
6	INF	INF	INF	INF	INF	INF	INF	7
7	INF	INF	INF	INF	INF	INF	INF	INF

Path Table

	0	1	2	3	4	5	6	7
0	-1	-1	0	0	0	-1	-1	-1
1	1	-1	-1	1	-1	-1	1	-1
2	2	2	-1	-1	-1	-1	2	-1
3	-1	-1	-1	-1	-1	-1	-1	3
4	-1	-1	4	-1	-1	-1	-1	4
5	-1	5	-1	5	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	6
7	-1	-1	-1	-1	-1	-1	-1	-1



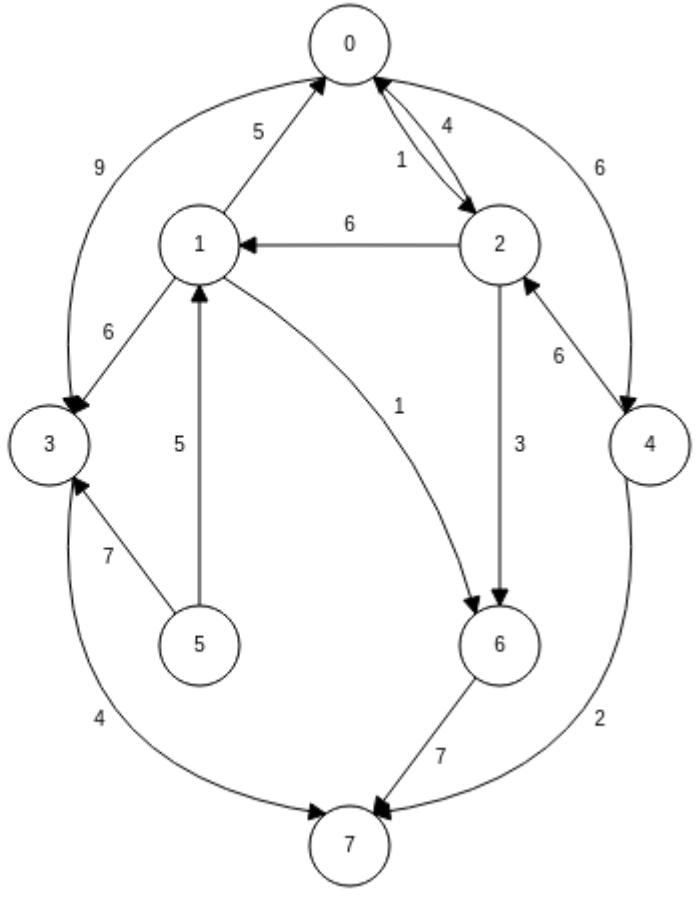
Floyd-Warshall All-Pairs Shortest Path

Cost Table

	0	1	2	3	4	5	6	7
0	INF	7	1	9	6	INF	4	8
1	5	INF	6	6	11	INF	1	8
2	4	6	INF	12	10	INF	3	10
3	INF	INF	INF	INF	INF	INF	INF	4
4	10	12	6	18	INF	INF	9	2
5	10	5	11	7	16	INF	6	11
6	INF	INF	INF	INF	INF	INF	INF	7
7	INF	INF	INF	INF	INF	INF	INF	INF

Path Table

	0	1	2	3	4	5	6	7
0	-1	2	0	0	0	-1	2	4
1	1	-1	0	1	0	-1	1	6
2	2	2	-1	1	0	-1	2	6
3	-1	-1	-1	-1	-1	-1	-1	3
4	2	2	4	1	-1	-1	2	4
5	1	5	0	5	0	-1	1	3
6	-1	-1	-1	-1	-1	-1	-1	6
7	-1	-1	-1	-1	-1	-1	-1	-1

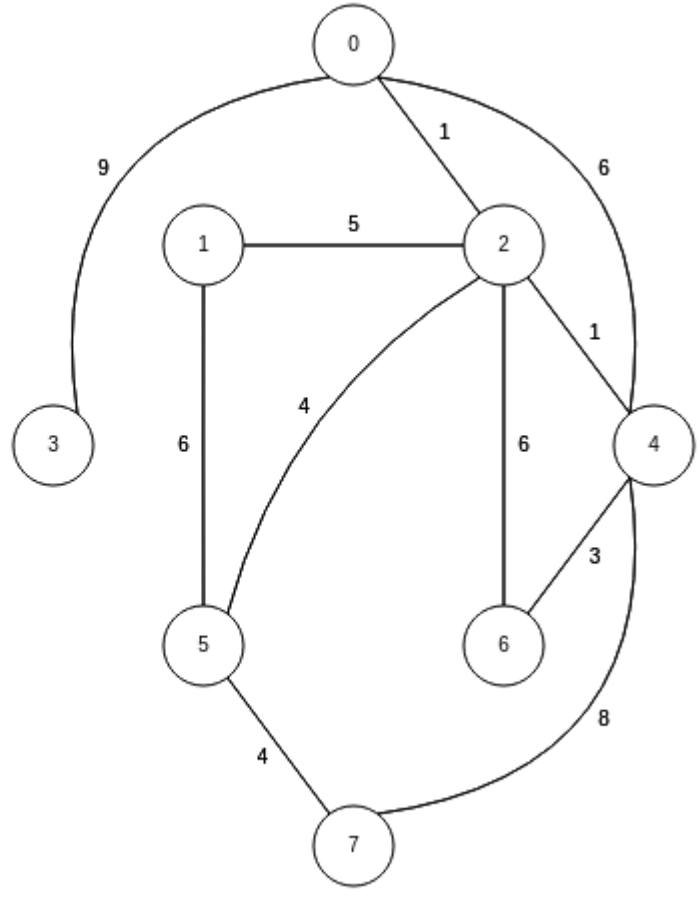


Floyd-Warshall

Kruskal Minimum Cost Spanning Tree

Disjoint Set

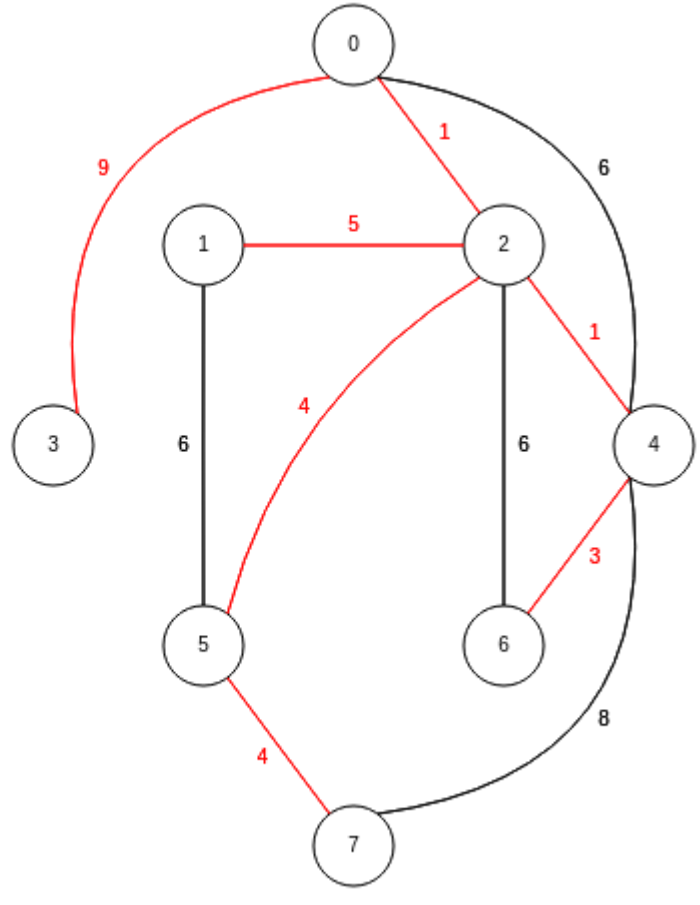
0	-1
1	-1
2	-1
3	-1
4	-1
5	-1
6	-1
7	-1



Kruskal Minimum Cost Spanning Tree

Disjoint Set

0	2
1	2
2	-8
3	2
4	2
5	2
6	2
7	2



Kruskal

Recursion

- Factorial
- Reversing a String
- N-Queens Problem

Factorial

```
def factorial(n):
    if (n <= 1):
        return 1
    else:
        subSolution = factorial(n - 1)
        solution = subSolution * n
        return solution
```

Factorial

```
def factorial(n):
    if (n <= 1):
        return 1
    else:
        subSolution = factorial(n - 1)
        solution = subSolution * n
        return solution

factorial(3) = 6
```

Factorial(3)

Reversing a String

```
def reverse(word):
    if (word == ""):
        return word
    else:
        subProblem = word[1:]
        subSolution = reverse(subProblem)
        solution = subSolution + word[0]
        return = solution
```

Reversing a String

```
def reverse(word):
    if (word == ""):
        return word
    else:
        subProblem = word[1:]
        subSolution = reverse(subProblem)
        solution = subSolution + word[0]
        return = solution

reverse(abc) = cba
```

Reverse("abc")

N-Queens Problem

```
def calcQueens(size):
    board = [-1] * size
   return  queens(board, 0, size)

def queens(board, current, size):
    if (current == size):
        return true
    else:
        for i in range(size):
            board[current] = i
            if (noConflicts(board, current):
                done = queens(board, current + 1, size)
                if (done):
                    return true
        return false

def noConflicts(board, current):
     for i in range(current):
       if (board[i] == board[current]):
          return false
       if (current - i == abs(board[current] = board[i])):
          return false
     return true
```

N-Queens Problem

```
def calcQueens(size):
    board = [-1] * size
   return  queens(board, 0, size)

def queens(board, current, size):
    if (current == size):
        return true
    else:
        for i in range(size):
            board[current] = i
            if (noConflicts(board, current):
                done = queens(board, current + 1, size)
                if (done):
                    return true
        return false

def noConflicts(board, current):
     for i in range(current):
        if (board[i] == board[current]):
            return false
        if (current - i == abs(board[current] = board[i])):
            return false
     return true
```

0	1	2	3
1	3	0	2

		Q	
Q			
			Q
	Q		

Qeens(4)

PDF Export

It's supported, just follow the reveal.js instructions

Each action (fragment) is exported in a separate page

A PDF export of this demo is available here