Algorithms, Data Structures

K-Digital Training

Data Structure & Algorithm

Algorithm

Algorithm

> Algorithm is a series of contained steps which you follow in order to achieve some goal, or to produce some output

> 목표를 달성하거나 결과물을 생산하기 위해 필요한 과정들

1 부터 100까지 더해보아요

누군가는

`1+2+3+4+...`

누군가는

```
1+100=101
2+99=101
...
101*50=5050
```

누군가는

```
n(n+1)/2
```

There are many ways to solve problem.

Algorithm is focus on clarity

Conditions

- has external input
- has 1 or more result
- clarity
- finite trial
- simplicity

Clarity

time complexity == big O notation

> 자료의 수 (n)이 증가할 때 시간의 증가 패턴을 나타낸 것

big O notation

1

log n

n

n log n

n^2

n^3

2^n

n!

O(1) : constant

- 값에 대한 키 또는 인덱스를 알고 있을 경우
  minsu_exam_result = {"kor":95,"math":40}

`minsu_exam_result['kor']`

```
result = 0
n = 100
result = n*(n+1)/2
```

O(log n) : logarithmic

- 배열에서 값을 접근할 때 앞 또는 뒤에서 접근 선택이 가능할 경우

```
animals = ['cat', 'dog', 'fox', 'giraffe', 'hippo', 'koyote', ..]
```

O(n): linear

- 자료의 수와 시도횟수가 1:1 관계인 경우

```
result = 0
for i in range(1,100+1):
        result += i
print(result)
```

O(n^2): quadratic

- 자료의 참조를 이중으로 하게 될 경우

```
result = 0
for i in range(1,10+1):
        for j in range(1,j+1):
                result += j
```

Sort algorithms

- O(n^2)
  - Bubble sort
  - Selection sort
  - Insertion sort
- O(n log n)
  - Merge sort
  - Heap sort
  - Quick sort

Bubble sort

https://www.youtube.com/embed/Cq7SMsQBEUw

- 1:1로 n(n-1)/2 번 수행하는 방법
- 최악..

Selection sort

https://www.youtube.com/embed/92BfuxHn2XE

- 가장 작은 값부터 순서대로 정렬
- 인간과 가장 가까운 정렬법

Insertion sort

https://www.youtube.com/embed/8oJS1BMKE64

- n번째 요소를 처음부터 n-1번째 까지 비교하면서 값을 끼워넣는 법
- 윗 방법들보단 빠름

Merge sort

https://www.youtube.com/embed/ZRPoEKHXTJg

6 5 3 1 8 7 2 4

- 두개씩 쪼개 각각을 비교하며 정렬하는 방법
- 데이터 상태에 큰 영향을 받지 않음

Heap sort

https://www.youtube.com/embed/_bkow6IykGM

6 5 3 1 8 7 2 4

- 데이터를 힙에 넣은 뒤 최대값(루트)을 출력하고 힙에서 제거

Quick sort

https://www.youtube.com/embed/8hEyhs3OV1w

6 5 3 1 8 7 2 4

- 피벗을 기준으로 큰값 작은 값을 나눈 뒤, 피벗을 옮겨 다시 수행하는 방법
- 평균적으로 가장 빠른 방법

Data Structure

Data Structure

> Data structure is a particular way of organizing data in a computer so that it can be used efficiently.

So, Data Structure is..

Data Structures in Web Development

Array & Hash(Dictionary) - indexing post

```
in RDB
[articleId, title, body, userId, view]
```

```
[{
        userId: 1,
        articleId: 1,
        view: 100,
        title: "sunt aut",
        body: "quia et suscipit suscipit"
},
...

]
```

Data Structures in Web Development

Tree - DOM rendering performance, reply

```
<html>
<head></head>
<body>
<h1></h1>
<p></p>
</body>
</html>
```

Data Structures in Web Development

- Binary Tree Search
  - Queue(BFS, Breadth First Search)
  - Stack(DFS, Depth First Search)
- Music Player, Image Viewer, undo/redo
  - Linked List

We'll Learn about..

- Stack
- Queue
- Linked List

Stack

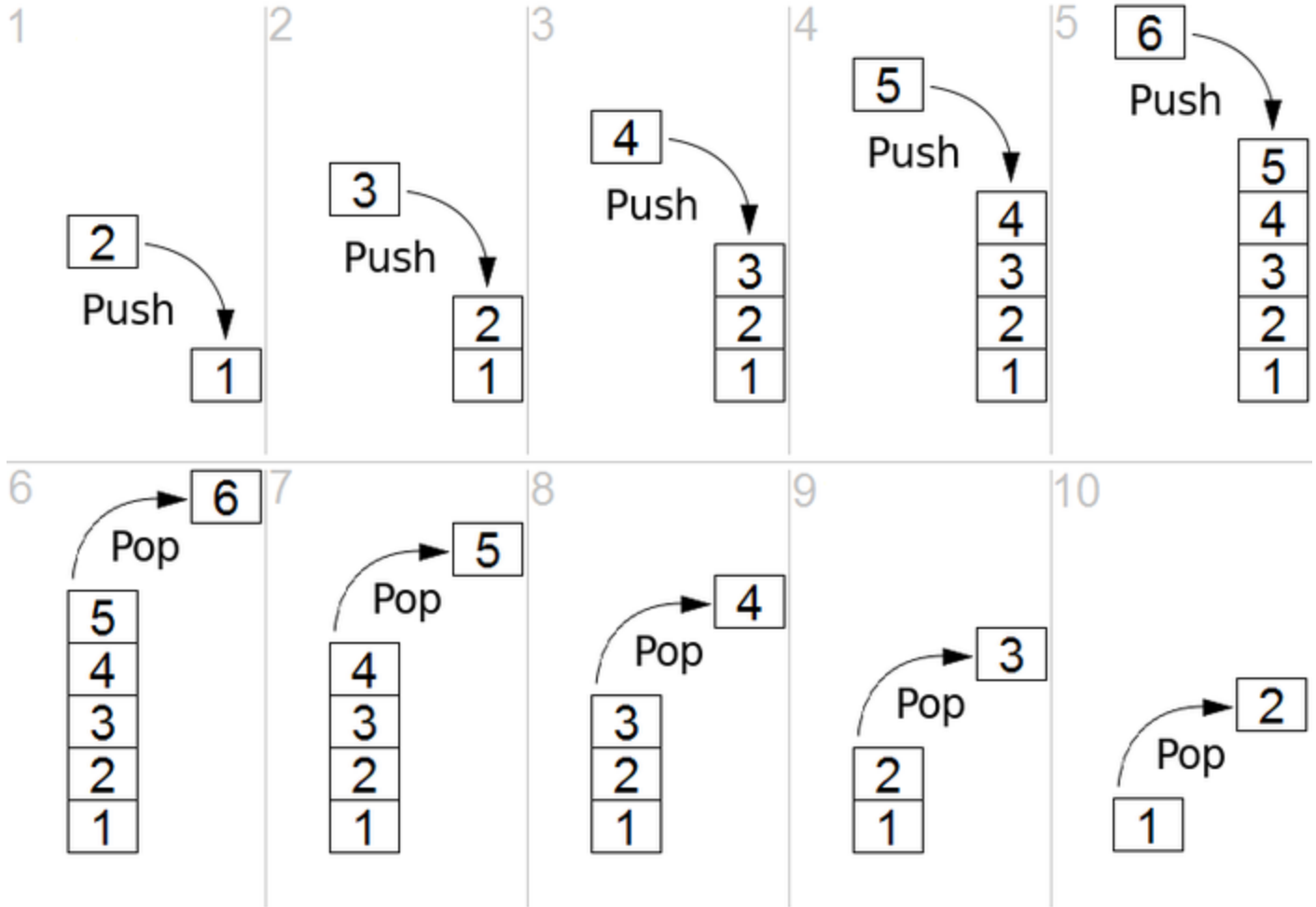
Stack

> A stack is an abstract data type that serves as a collection of elements, with two principal operations.

- push: which adds an element to the collection
- pop: which removes the most recently added element that was not yet removed

LIFO

> Last In, First Out


1
2
Push
1
2
3
Push
2
1
3
4
Push
3
2
1
4
5
Push
4
3
2
1
5
6
Push
5
4
3
2
1
6
7
8
9
10
6
Pop
5
4
3
2
1
5
Pop
4
3
2
1
4
Pop
3
2
1
3
Pop
2
1
2
Pop
1

Let's Create Stack class

```
function Stack() {
        //properties, methods
        var items = [];
}
```

push & pop

```
function Stack( ) {
        //properties, methods
        var items = [];

        this.push = function(element){
                return items.push(element);
        };


        this.pop = function( ){
                return items.pop( );
        };
}
```

peek & isEmpty

```
function Stack() {
        //underneath push & pop

...

        this.peek = function(){
                return items[items.length-1];
        };
        this.isEmpty = function(){
                return items.length == 0;
        };


}
```

size & clear & print

```
function Stack() {
        //underneath peek & isEmpty

...

        this.size = function(){
                return items.length;
        };

        this.clear = function(){
                items = [];
        };

        this.print = function(){
                console.log(items.toString());
        };

}
```

Let's push with Stack class

```
> var stack = new Stack();
> console.log(stack.isEmpty());

> stack.push(5);
> stack.push(2);
> stack.push(8);

> console.log(stack.peek());

> stack.push(11);

> console.log(stack.size());
> console.log(stack.isEmpty());
```

Let's pop with Stack class

```
> stack.pop( );
> stack.pop( );

> console.log(stack.size( ));
> stack.print( );
```

Queue

Queue

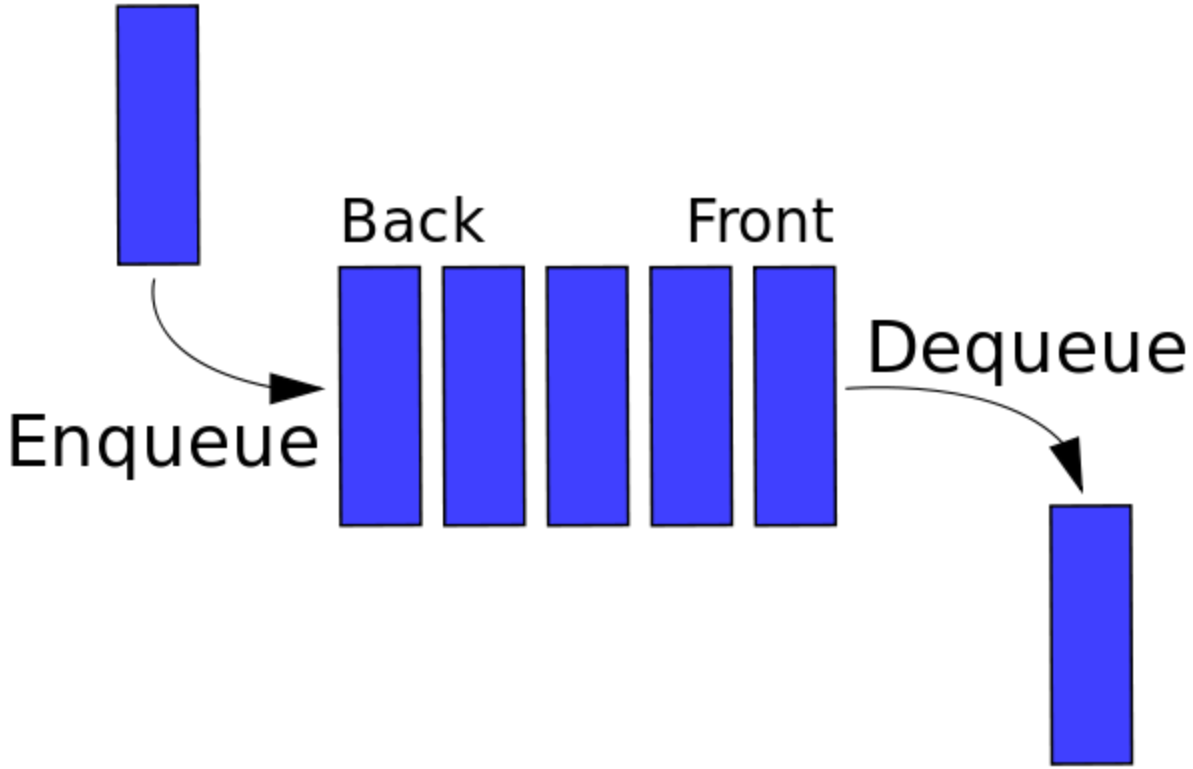
> a queue is a particular kind of abstract data type or collection in which the entities in the collection are kept in order.

FIFO

First In First Out

Enqueue & Dequeue

- Enqueue: addition of entities to the rear terminal position
- Dequeue: removal of entities from the front terminal position


Back
Front
Enqueue
Dequeue

Let's Create Queue class

```
function Queue() {
        //properties, methods
        var items = [];
}
```

Enqueue & Dequeue

```
function Queue() {
        //properties, methods
        this.enqueue = function(element) {
                items.push(element);
        };
        this.dequeue = function() {
                return items.shift();
        };
}
```

front & isEmpty

```
function Queue() {
        //underneath Enqueue & Dequeue

...

        this.front = function() {
                return items[0];
        };
        this.isEmpty = function() {
                return items.length == 0;
        };
}
```

clear & size & print

```
function Queue() {
        //underneath front & isEmpty

...

        this.clear = function() {
                items = [];
        };
        this.size = function() {
                return items.length;
        };
        this.print = function() {
                console.log(items.toString());
        };

}
```

Let's Enqueue with Queue class

```
> var queue = new Queue();
> console.log(queue.isEmpty());

> queue.enqueue("Fast");
> queue.enqueue("Campus");
> queue.enqueue("School");

> queue.print();
> console.log(queue.size());
> console.log(queue.isEmpty());
```

Let's Dequeue with Queue class

```
> queue.dequeue();
> queue.dequeue();

> queue.print();
```

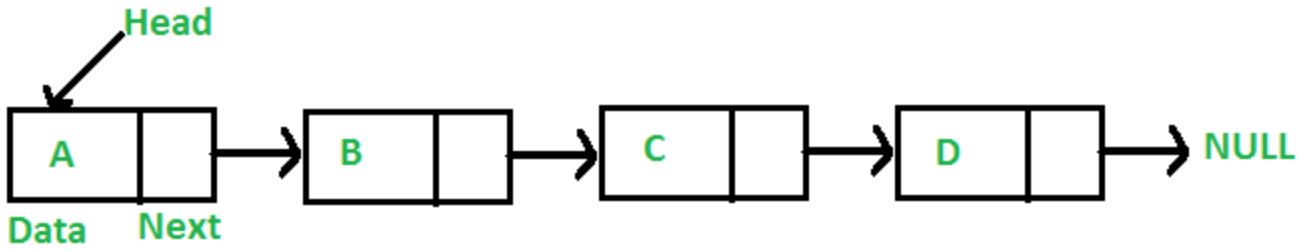
Linked List

Linked List

> A linked list is a linear collection of data elements, in which linear order is not given by their physical placement in memory.

Linked List

- Can be used to store linear data of simlar types.



Create Node Class

```
function Node(element) {
        this.element = element;
        this.next = null;
}
```

Create Linked List Class

```
function LinkedList(){
        this.head = new Node("head");
        this.find = find;
        this.insert = insert;
        this.remove = remove;
        this.display = display;
}
```

To insert,

```
function find(item){
        var currNode = this.head;
        while (currNode.element != item){
                currNode = currNode.next;
        }
        return currNode;
}
```

```
function insert(newElement, item) {
        var newNode = new Node(newElement);
        var current = this.find(item);
        newNode.next = current.next;
        current.next = newNode;
}
```

display the elements

```
function display() {
        var currNode = this.head;
        while(!(currNode.next==null)){
                console.log(currNode.next.element);
                currNode=currNode.next;
        }
}
```

```
var countries = new LinkedList();
countries.insert("Seoul", "head");
countries.insert("Incheon", "Seoul");
countries.insert("Daejeon", "Incheon");
countries.display();
```

Remove Node

```
function findPrevious(item){
        var currNode = this.head;
        while(!(currNode.next==null)&&(currNode.next.element!=item)){
                currNode = currNode.next;
        }
        return currNode;
}
```

```
function remove(item) {
        var prevNode = this.findPrevious(item);
        if(!(prevNode.next==null)){
                prevNode.next = prevNode.next.next;
        }
}
```

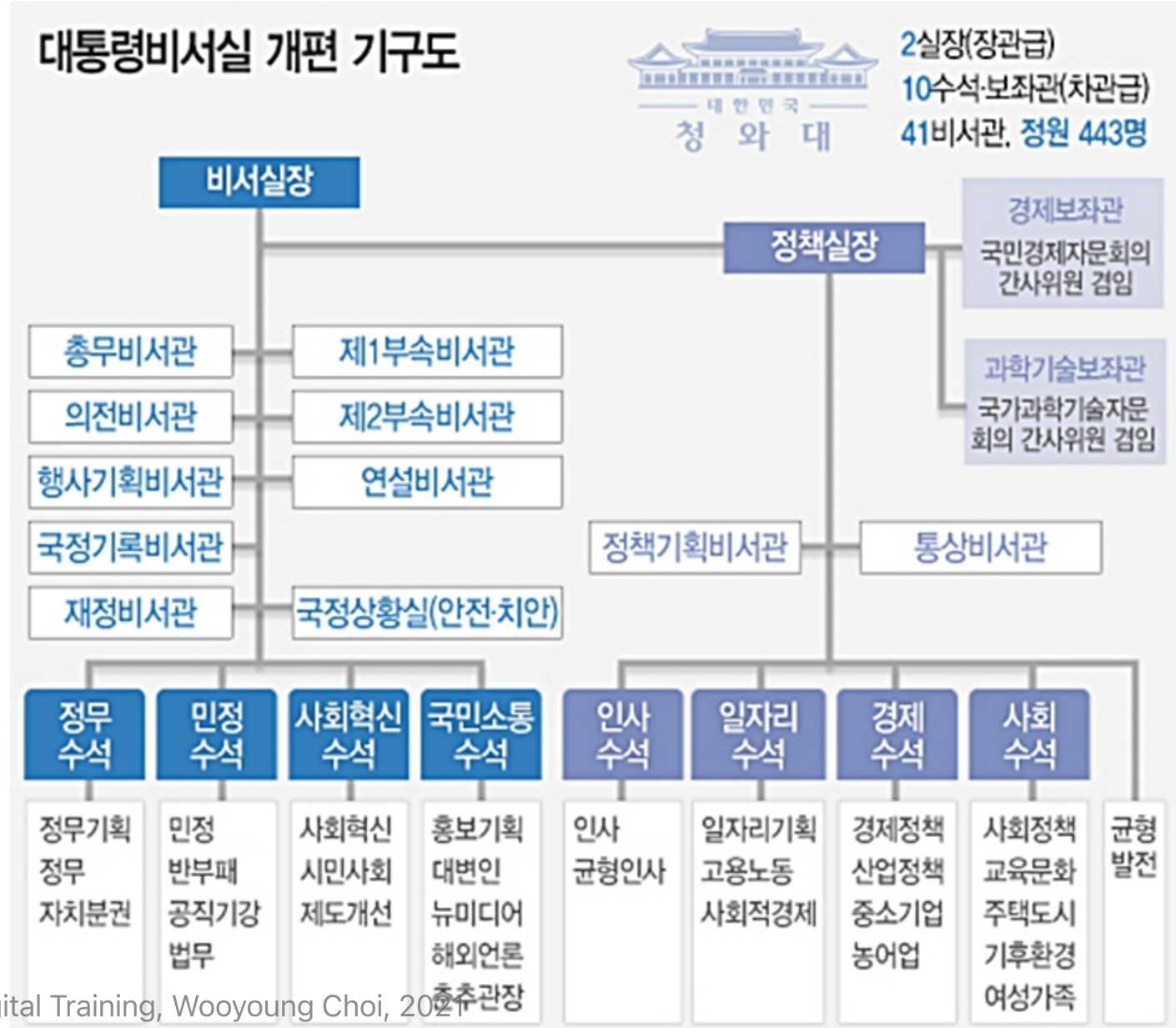
Tree

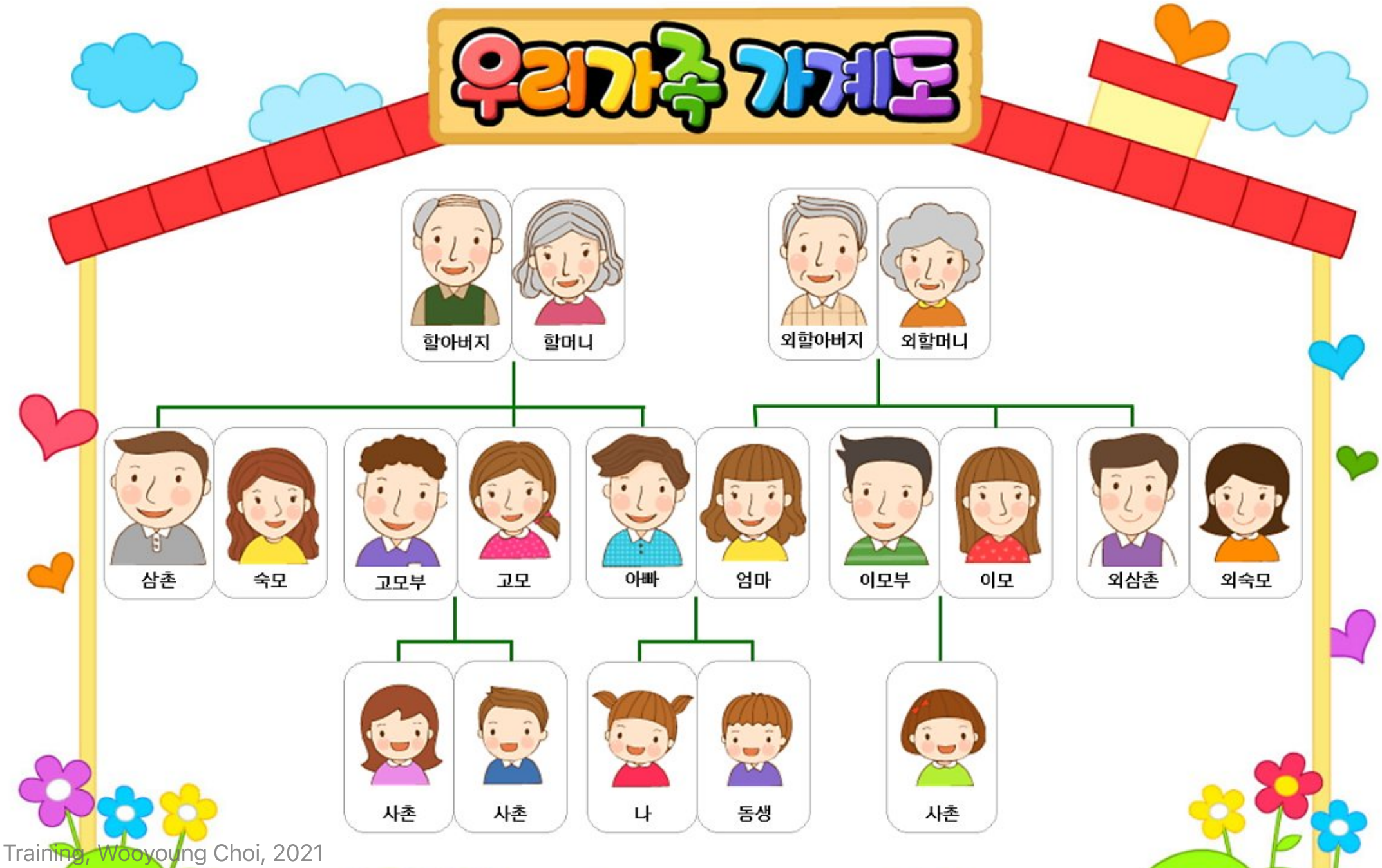
Tree

> A tree is an abstract model of a hierarchical structure.

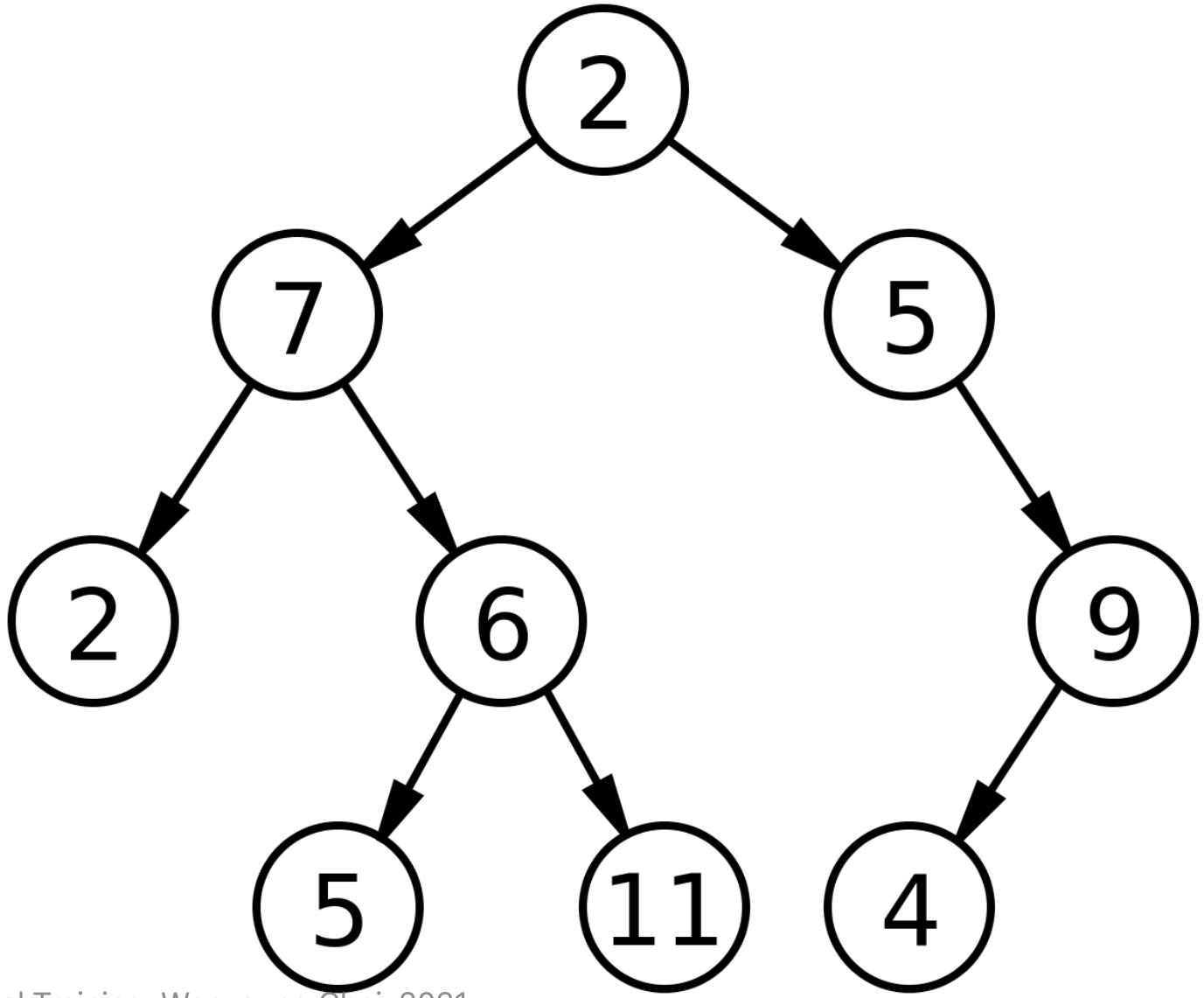
- hierarchical: arranged in order of rank.

Tree


대통령비서실 개편 기구도
대 한 민 국
청 와 대
2실장(장관급)
10수석·보좌관(차관급)
41비서관, 정원 443명
비서실장
정책실장
경제보좌관
국민경제자문회의 간사위원 겸임
과학기술보좌관
국가과학기술자문회의 간사위원 겸임
총무비서관
제1부속비서관
의전비서관
제2부속비서관
행사기획비서관
연설비서관
국정기록비서관
재정비서관
국정상황실(안전·치안)
정책기획비서관
통상비서관
정무 수석
민정 수석
사회혁신 수석
국민소통 수석
인사 수석
일자리 수석
경제 수석
사회 수석
정무기획
정무
자치분권
민정
반부패
공직기강
법무
사회혁신
시민사회
제도개선
홍보기획
대변인
뉴미디어
해외언론
춘추관장
인사
균형인사
일자리기획
고용노동
사회적경제
경제정책
산업정책
중소기업
농어업
사회정책
교육문화
주택도시
기후환경
여성가족
균형발전


우리가족 가계도
할아버지
할머니
외할아버지
외할머니
삼촌
숙모
고모부
고모
아빠
엄마
이모부
이모
외삼촌
외숙모
사촌
사촌
나
동생
사촌

Tree



Tree

- root: 2
- level: (0 ~ 3)
- child of 2: 7,5
- subtree: 6,5,11
- Node: (9)
- edge: (8)

Binary Search Tree

> A node in a binary tree has at most two children: left child, right child

- if root == null, node = newNode
- left child < right child

Create Stack(), QueueWithStack()

- 3 members 1 repo
- pull 받아 사용해야 하며, 피처 브랜치에서 작업해야 합니다.
- Requirement
  - stack
    - push, pop
    - peek, isEmpty
    - size, print
  - queue with stack
    - enqueue, dequeue

Create Queue with 2 Stacks

```
function Queue_with_stack() {
    var inBox = [];
    var outBox = [];

    this.enqueue = function(num) {
        inBox.push(num);
    };

    this.dequeue = function() {
        if (outBox.length > 0) {
            return outBox.pop();
        }

        while(inBox.length > 1) {
            outBox.push(inBox.pop());
        }

        return inBox.pop();
    };
}
```