Fastcampus

Web Programming & Frontend Dev SCHOOL

Data Structures, Gulp(2)

Data Structure

Data Structure

- Linked List
- Tree

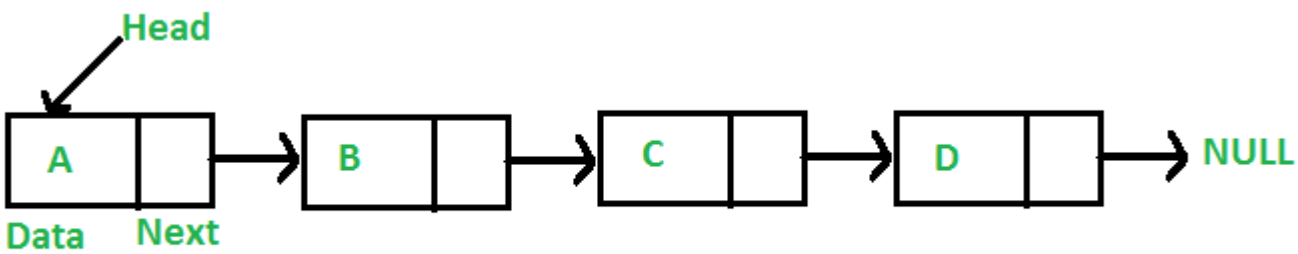
Linked List

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.

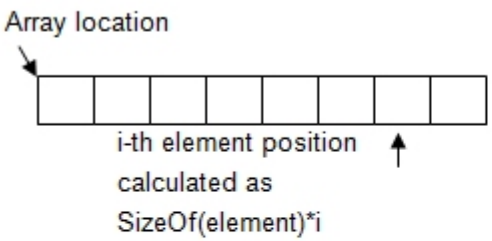


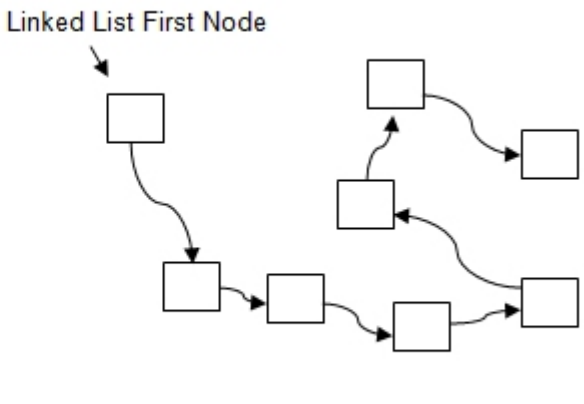
Array를 **놔두고 굳이 왜???**

Array

```
var myArray = [];
```

Array vs Linked List





Array vs Linked List

Array	vs	Linked List
Fixed	Size	Dynamic
Hard	Insert	Easy
Hard	Deletion	Easy
Allowed	Random Access	Not allowed
doesn't need	Extra memory space	required

Array vs Linked List

Array	vs	Linked List
o(1)	access	o(n)
o(n)	search	o(n)
o(n)	insert	o(1)
o(n)	remove	o(1)

Let's Create Linked List

```
function LinkedList(){
    //define Node
    var Node = function(element){
        this.element = element;
        this.next = null;
    };

...

}
```

Linked List

```
function LinkedList(){
    //beneath Node

...

    var length = 0;
    var head = null;

    this.append = function(element){};
    this.insert = function(position, element){};
    this.removeAt = function(position){};
    this.remove = function(element){};

...

}
```

Linked List

- append: add new item to the end of the list
- insert: insert new item at a specified position in the list
- remove: removes an item from the list
- removeAt: removes an item from a specified in the list

Linked List

```
function LinkedList(){
    //beneath remove

...

    this.indexOf = function(element){};
    this.isEmpty = function(){};
    this.size = function(){};
    this.toString = function(){};
    this.print = function(){};
}
```

Linked List

- indexOf: returns the index of the element in the list

Linked List - append

```
this.append = function(element){
    var note = new Node(element),
        current;
    if (head === null){
        head = node;
    } else {
        current = head;
        while(current.next){
            current = current.next;
        }
    }

    length++;
};
```

Linked List - removeAt

```
this.removeAt = function(position){
    if (position > -1 && position < length){
        var current = head,
            previous,
            index = 0;

        if (position === 0) {
            head = current.next;
        } else {
            while (index++ < position) {
                previous = current;
                current = current.next;
            }
            previous.next = current.next;
        }
        length--;

        return current.element;
    } else {
        return null;
    }
};
```

Linked List - insert

```
this.insert = function(position, element){
    if (position >= 0 && position <= length){
        var node = new Node(element),
            current = head,
            previous,
            index = 0;
...
```

Linked List - insert

```
...
        if (position === 0) {
            node.next = current;
            head = node;
        } else {
            while (index++ < position) {
                previous = current;
                current = current.next;
            }
            node.next = current;
            previous.next = node;
        }
        length++;
        return true;

    } else {
        return false;
    }
};
```

Linked List - remove

```
...
        this.remove = function(element) {
            var index = this.indexOf(element);
            return this.removeAt(index);
        }
```

Linked List - indexOf

```
this.indexOf = function(element){
    var current = head,
        index = -1;

    while (current){
        if (element === current.element) {
            return index;
        }
        index++;
        current = current.next;
    }

    return -1;
};
```

Linked List - isEmpty, size, toString, getHead

```
this.isEmpty = function(){
    return length === 0;
};

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

this.toString = function(){
    var current = head,
        string = '';

    while (current) {
        string = current.element;
        current = current.next;
    }
    return string;
};

this.getHead = function(){
    return head;
};
```

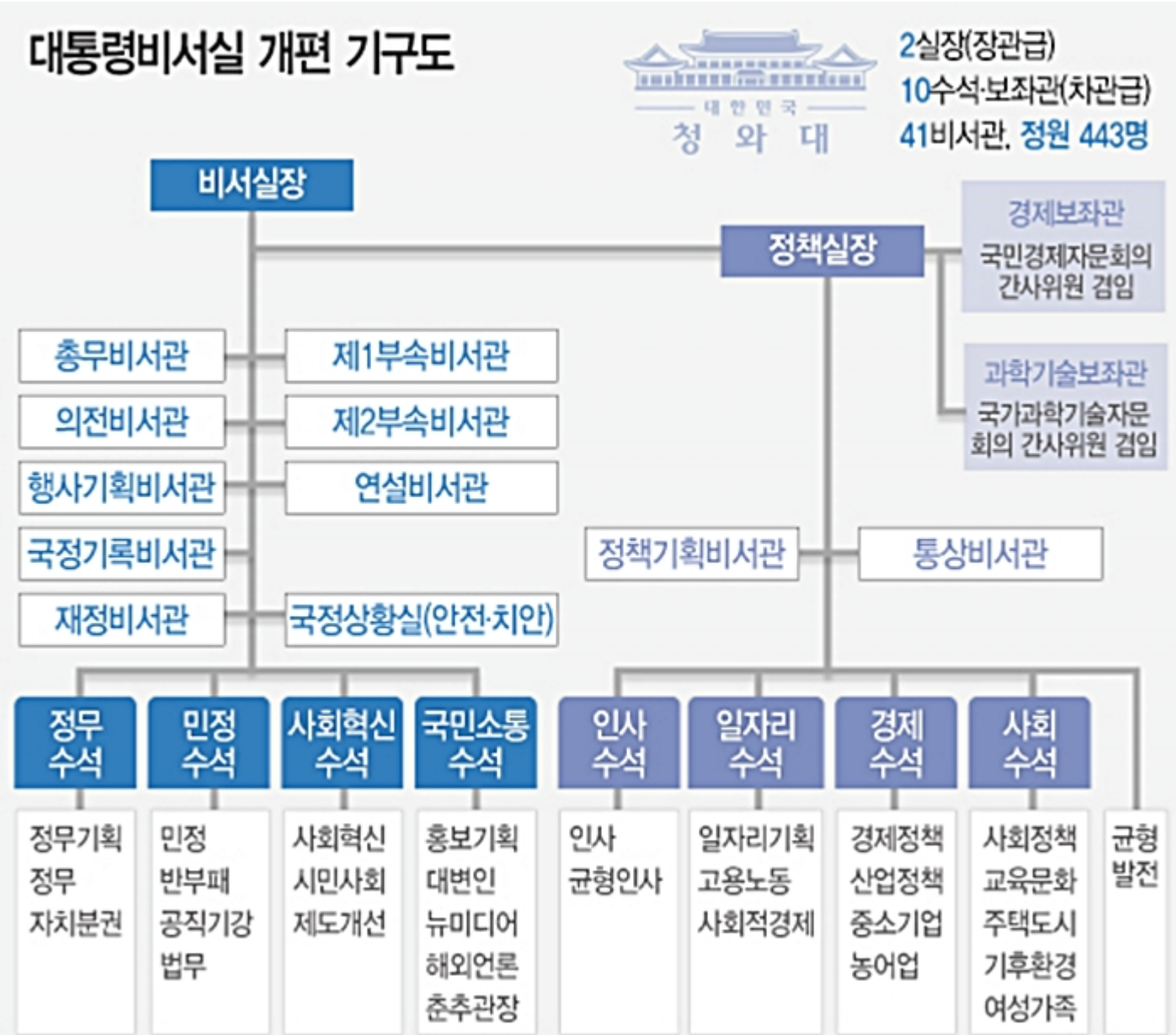
Tree

Tree

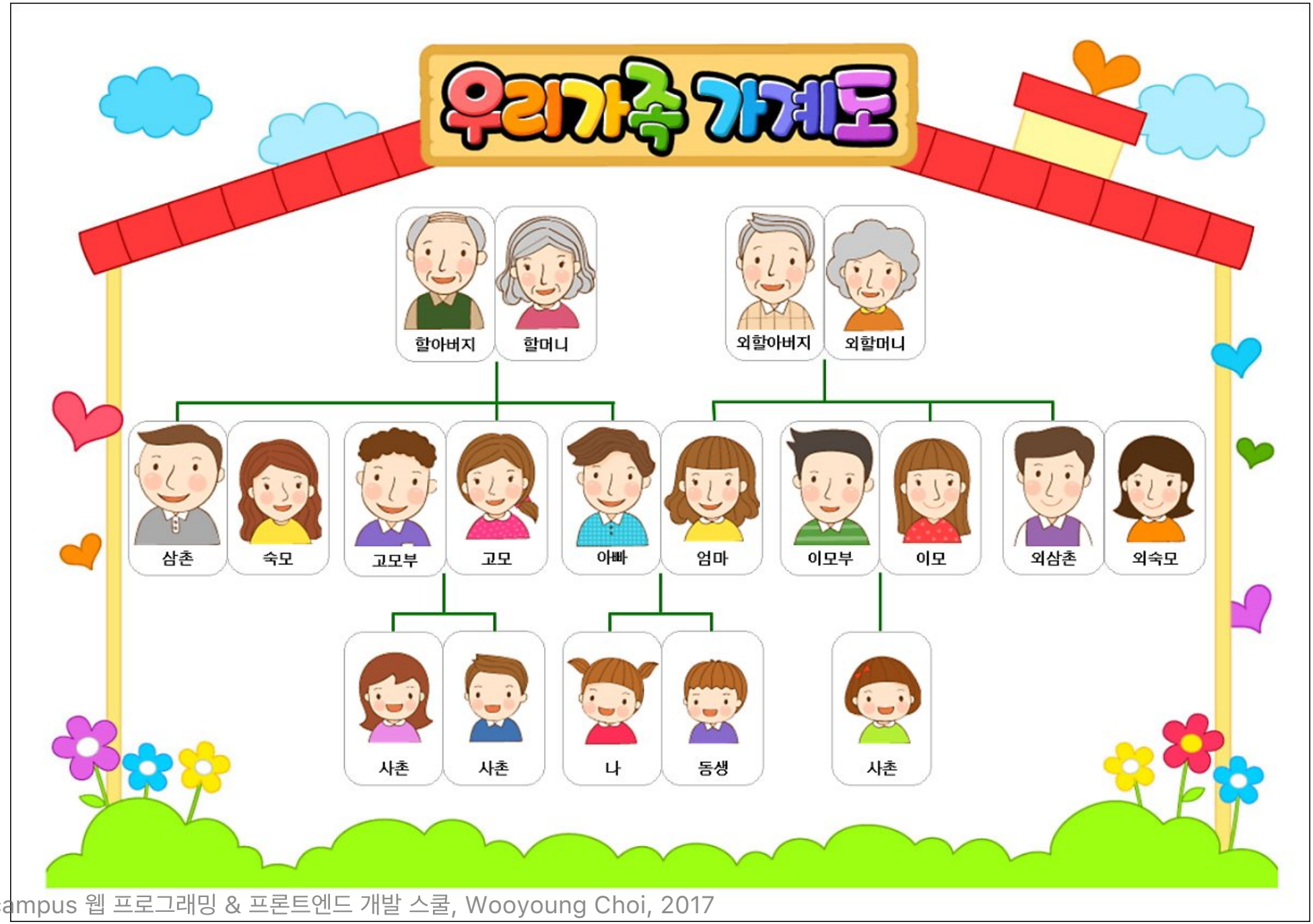
> A tree is an abstract model of a hierarchical structure.

- hierarchical: arranged in order of rank.

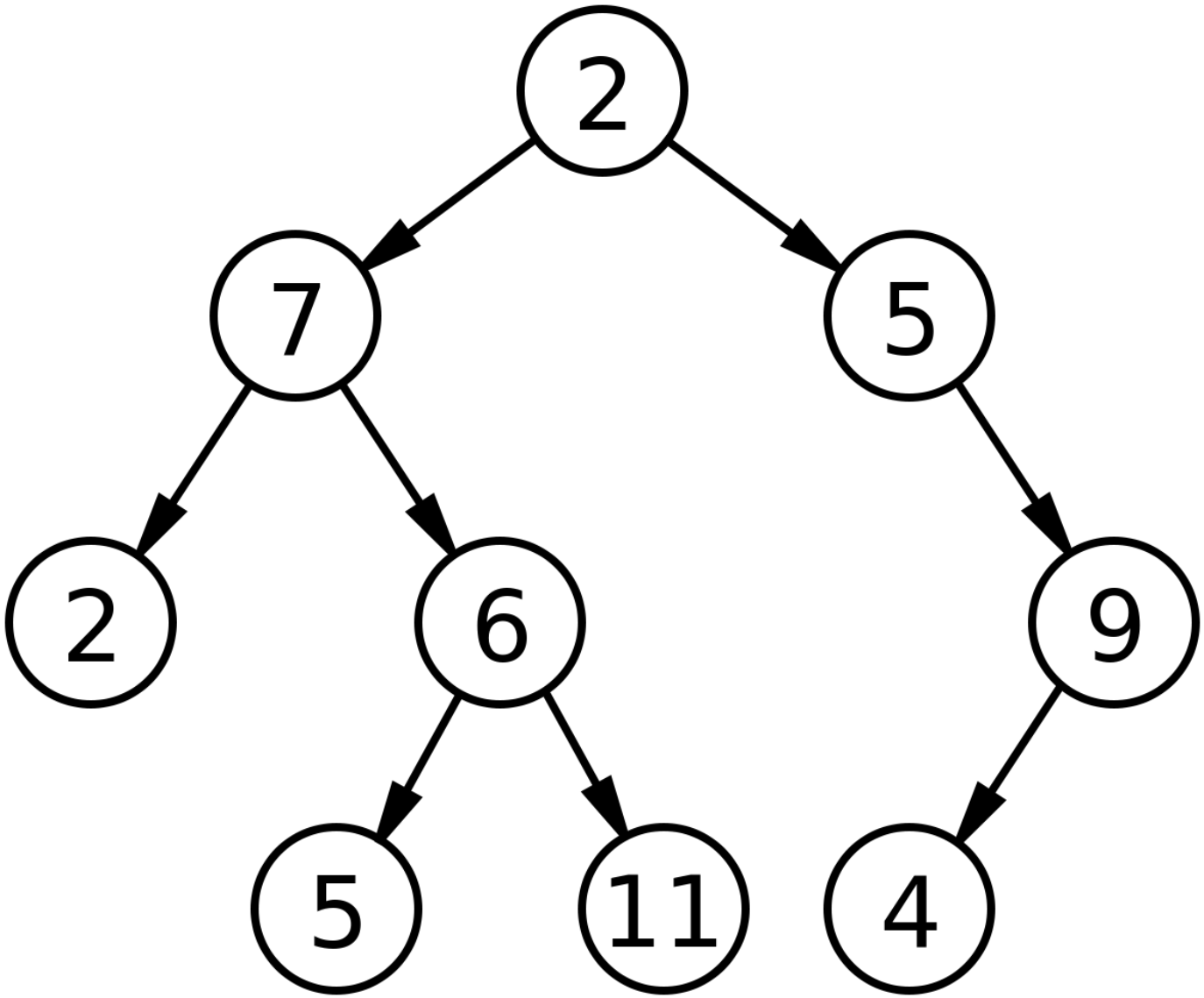
Tree


대통령비서실 개편 기구도
대한민국
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
균형 발전

Tree


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

Binary Search Tree

```
function BinarySearchTree() {
    var Node = function(key) {
        this.key = key;
        this.left = null;
        this.right = null;
    };

    var root = null;

...

}
```

Binary Search Tree - insert

```
...
    this.insert = function(key){
        var newNode = new Node(key);

        if (root === null){
            root = newNode;
        } else {
            insertNode(root, newNode);
                }
        };

...
```

Binary Search Tree - insertNode

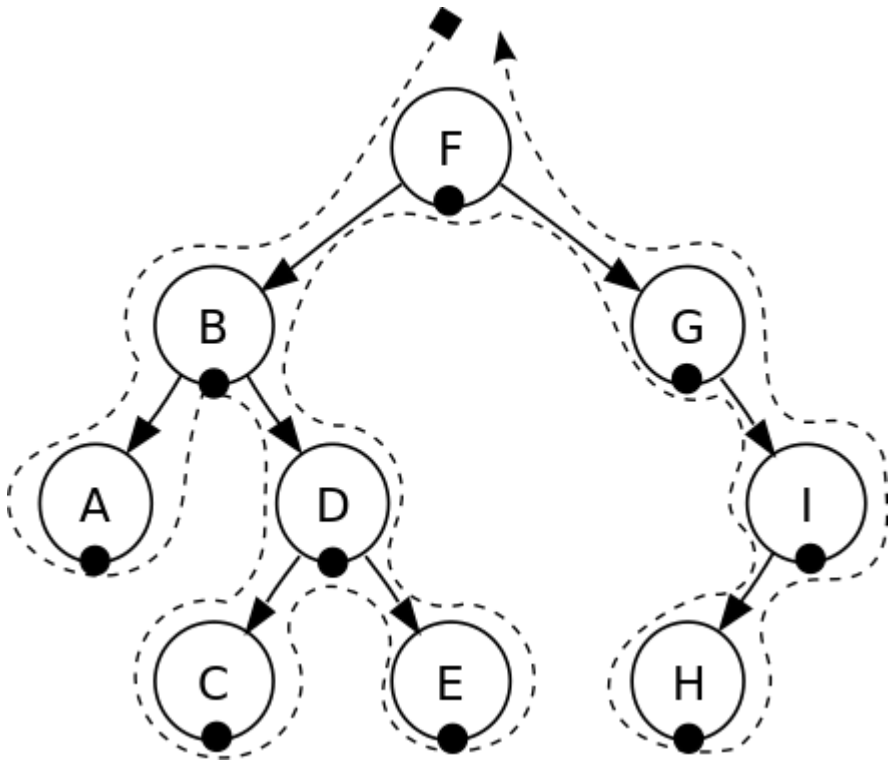
```
...
    var insertNode = function(node, newNode){
        if (newNode.key < node.key){
            if (node.left === null){
                node.left = newNode;
            } else {
                insertNode(node.left, newNode);
            }
        } else {
            if (node.right === null){
                node.right = newNode;
            } else {
                insertNode(node.right, newNode);
            }
        }
    };
```

Binary Search Tree - insert

```
var tree = new BinarySearchTree();
```

`[11,7,15,5,3,9,8,10,13,12,14,20,18,25] + [6]`

Binary Search Tree - inOrderTraverse



Binary Search Tree - inOrderTraverse

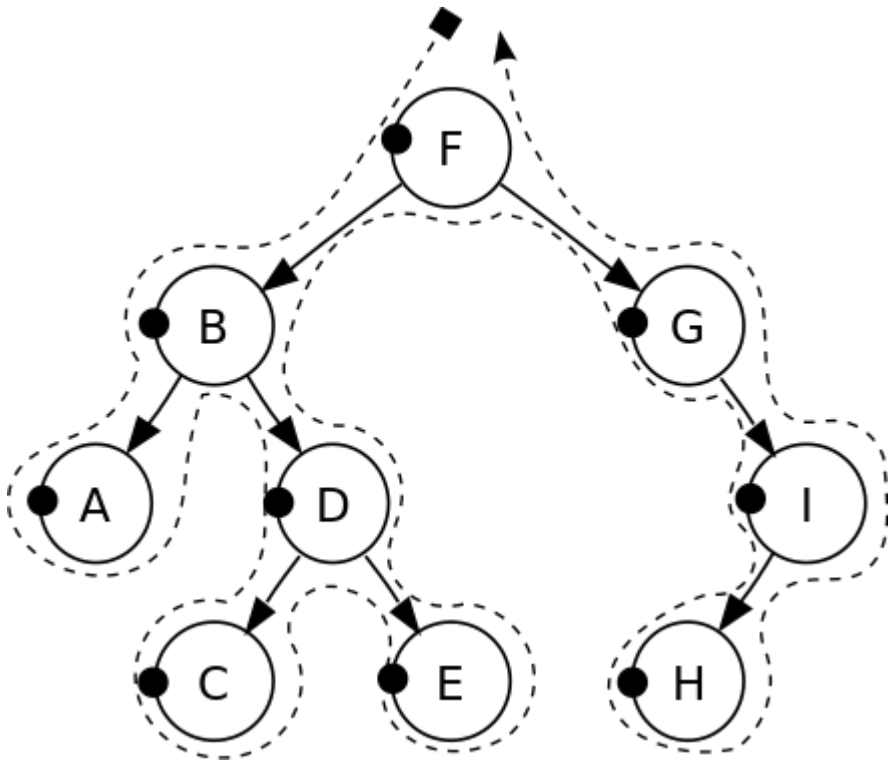
```
...
    this.inOrderTraverse = function(callback){
        inOrderTraverseNode(root, callback);
    };

    var inOrderTraverseNode = function(node, callback){
        if (node !== null){
            inOrderTraverseNode(node.left, callback);
            callback(node.key);
            inOrderTraverseNode(node.right, callback);
        }
    };
```

Binary Search Tree - printNode

```
function printNode(value) {
    console.log(value);
}
```

Binary Search Tree - preOrderTraverse

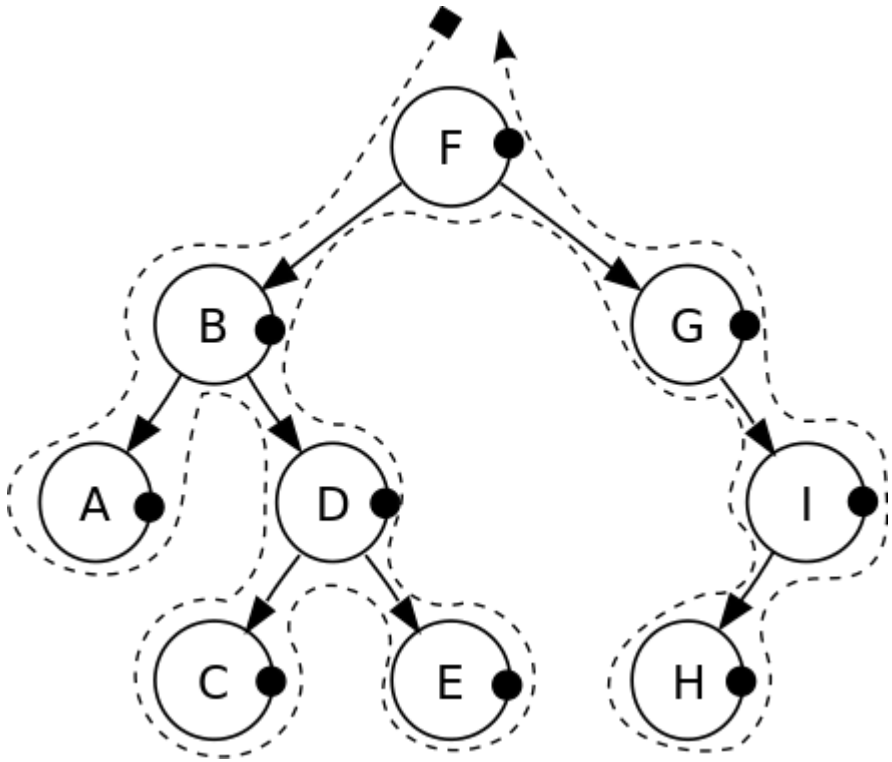


Binary Search Tree - preOrderTraverse

```
...
    this.preOrderTraverse = function(callback){
        preOrderTraverseNode(root, callback);
    };

    var preOrderTraverseNode = function(node, callback){
        if (node !== null){
            callback(node.key);
            preOrderTraverseNode(node.left, callback);
            preOrderTraverseNode(node.right, callback);
        }
    };
```

Binary Search Tree - postOrderTraverse



Binary Search Tree - postOrderTraverse

```
this.postOrderTraverse = function(callback){
    postOrderTraverseNode(root, callback);
};

var postOrderTraverseNode = function(node, callback){
    if (node !== null){
        postOrderTraverseNode(node.left, callback);
        postOrderTraverseNode(node.right, callback);
        callback(node.key);
    }
};
```

Binary Search Tree - find min value

```
this.min = function(){
    return minNode(root);
};

var minNode = function(node){
    if (node){
        while (node && node.left !== null){
            node = node.left;
        }

        return node.key;
    }
    return null;
};
```

Binary Search Tree - find max value

```
this.max = function(){
    return maxNode(root);
};

var maxNode = function(node){
    if (node){
        while (node && node.right !== null){
            node = node.right;
        }

        return node.key;
    }
    return null;
};
```

Binary Search Tree - find specific value

```
    this.search = function(key){
        return searchNode(root, key);
    };

    var searchNode = function(node, key){
        if (node == null){
            return false;
        }
        if (key < node.key){
            return searchNode(node.left, key);
        } else if (key > node.key){
            return searchNOde(node.right, key);
        } else {
            return true;
        }
    };
```

```
console.log(tree.search(1) ? 'Key 1 found.' : 'Key 1 not found.');
```

Conditional Operator

```
condition ? expr1 : expr2
```

MDN Docs

gulp

gulp-imagemin : image minify

```
gulp.task("imagemin", function(){
        pump([
                gulp.src(publicPath.src + 'img/*.jpg'),
                imagemin(),
                gulp.dest(publicPath.dest + 'img/')
        ]);
});
```

css minify(gulp-clean-css) : css minify

```
gulp.task("cleancss", function(){
        pump([
                gulp.src(publicPath.src + 'css/minify.css'),
                cleancss(),
                gulp.dest(publicPath.dest + 'css/')
        ]);
});
```

gulp-sass : convert .scss to .css

```
gulp.task("sass", function(){
        pump([
                gulp.src(publicPath.src + 'sass/*.scss'),
                sass().on('error', sass.logError),
                gulp.dest(publicPath.dest + 'css/')
        ]);
});
```

gulp-concat-css : concatenate css files

```
gulp.task("concatcss", function(){
        pump([
                gulp.src([publicPath.src + 'css/concat1.css', pu
                concat('concatenated.css'),
                gulp.dest(publicPath.dest + 'css/')
        ]);
});
```

clean(del)

```
gulp.task("clean", function(){
        return del.sync([publicPath.dest + 'js/*.js', publicPath
});
```

watch

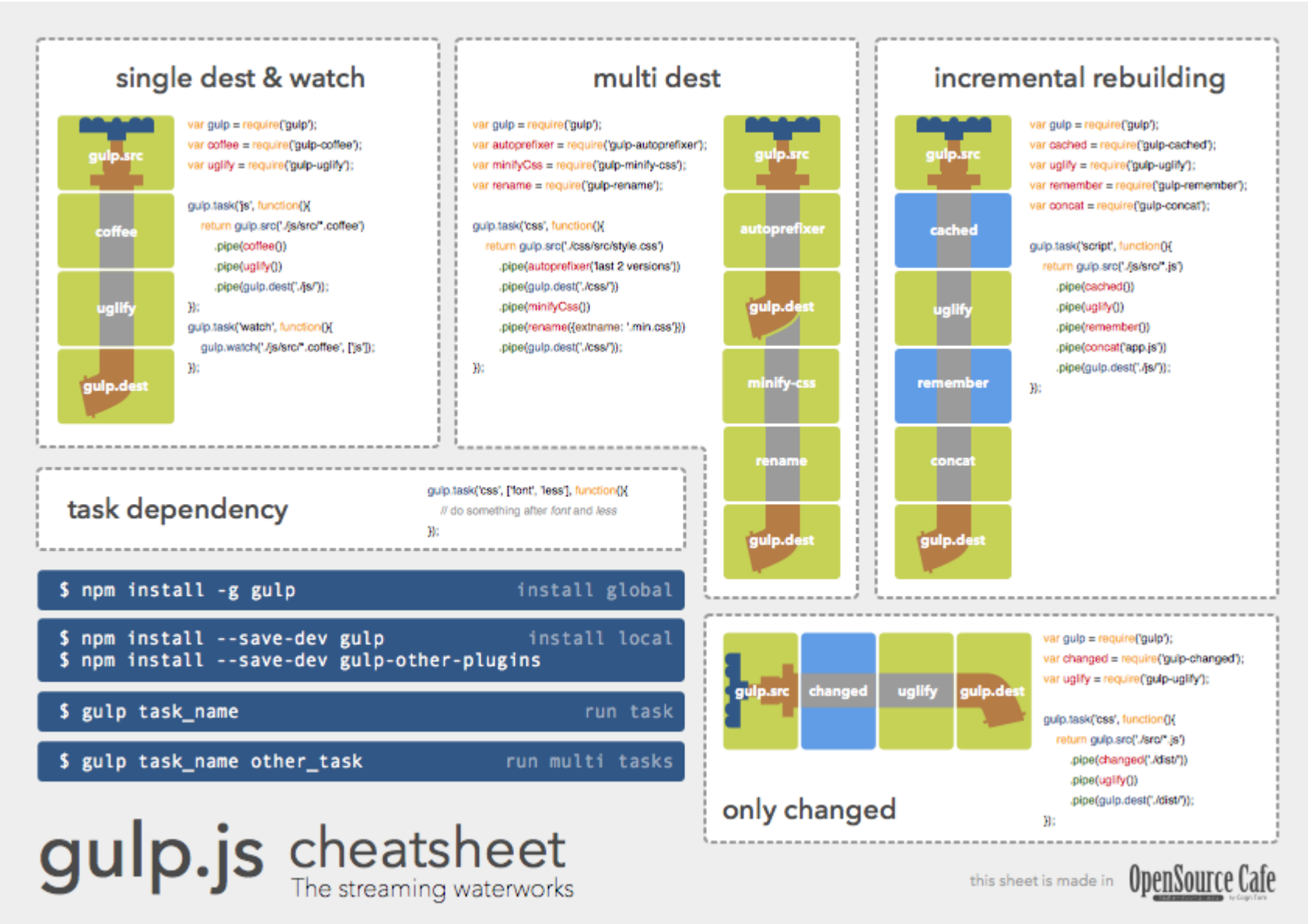
```
gulp.task("watch", function(){
        gulp.watch("public/src/*.js", ["uglify"]);
});

gulp.task("default", ["uglify", "watch"]);
```

watch

```
gulp.task("watch", function(){
        gulp.watch(publicPath.src + 'js/*.js', ["uglify", "conca
        gulp.watch(publicPath.src + 'css/*.css', ["cleancss", "c
        gulp.watch(publicPath.src + 'img/*.jpg', ["imagemin"]),
        gulp.watch(publicPath.src + 'sass/*.scss', ["sass"])
 +});
```

이외에도..


single dest & watch
gulp.src
coffee
uglify
gulp.dest
var gulp = require('gulp');
var coffee = require('gulp-coffee');
var uglify = require('gulp-uglify');
gulp.task('js', function(){
return gulp.src('./js/src/*.coffee')
.pipe(coffee())
.pipe(uglify())
.pipe(gulp.dest('./js/'));
});
gulp.task('watch', function(){
gulp.watch('./js/src/*.coffee', ['js']);
});
multi dest
var gulp = require('gulp');
var autoprefixer = require('gulp-autoprefixer');
var minifyCss = require('gulp-minify-css');
var rename = require('gulp-rename');
gulp.task('css', function(){
return gulp.src('./css/src/style.css')
.pipe(autoprefixer('last 2 versions'))
.pipe(gulp.dest('./css/'))
.pipe(minifyCss())
.pipe(rename({extname: '.min.css'}))
.pipe(gulp.dest('./css/'));
});
gulp.src
autoprefixer
gulp.dest
minify-css
rename
gulp.dest
incremental rebuilding
gulp.src
cached
uglify
remember
concat
gulp.dest
var gulp = require('gulp');
var cached = require('gulp-cached');
var uglify = require('gulp-uglify');
var remember = require('gulp-remember');
var concat = require('gulp-concat');
gulp.task('script', function(){
return gulp.src('./js/src/*.js')
.pipe(cached())
.pipe(uglify())
.pipe(remember())
.pipe(concat('app.js'))
.pipe(gulp.dest('./js/'));
});
task dependency
gulp.task('css', ['font', 'less'], function(){
// do something after font and less
});
$ npm install -g gulp install global
$ npm install --save-dev gulp install local
$ npm install --save-dev gulp-other-plugins
$ gulp task_name run task
$ gulp task_name other_task run multi tasks
gulp.src
changed
uglify
gulp.dest
var gulp = require('gulp');
var changed = require('gulp-changed');
var uglify = require('gulp-uglify');
gulp.task('css', function(){
return gulp.src('./src/*.js')
.pipe(changed('./dist/'))
.pipe(uglify())
.pipe(gulp.dest('./dist/'));
});
only changed
gulp.js cheatsheet
The streaming waterworks
this sheet is made in OpenSource Cafe

챌린지!!

40라인으로 슬랙봇만들기 를 이해하고 커스터마이즈 할 수 있다면 여러분은 node.js express.js heroku git REST api를 이해하셨습니다.