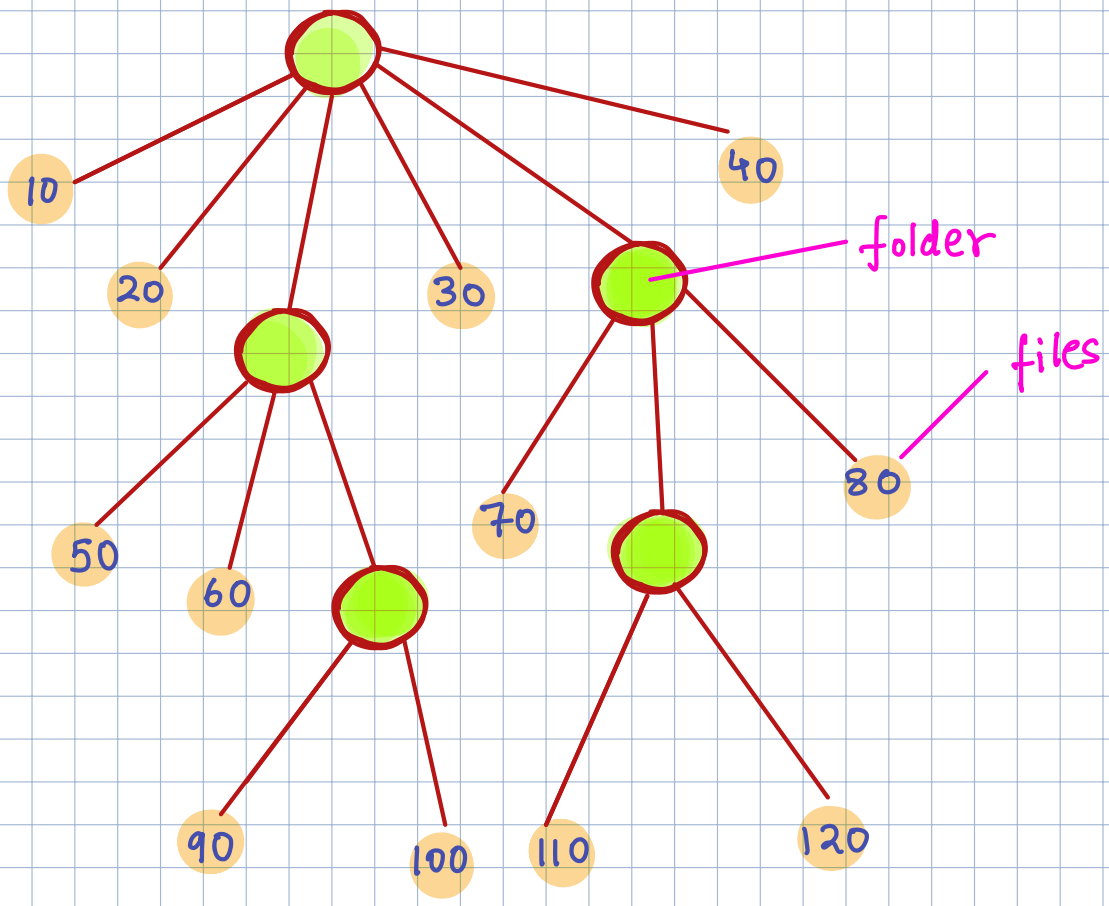
# ARRAY FLAT




Representation of tree in the form of array

Opening square bracket:- create folder and go one level down

closing square bracket :- go one level up.

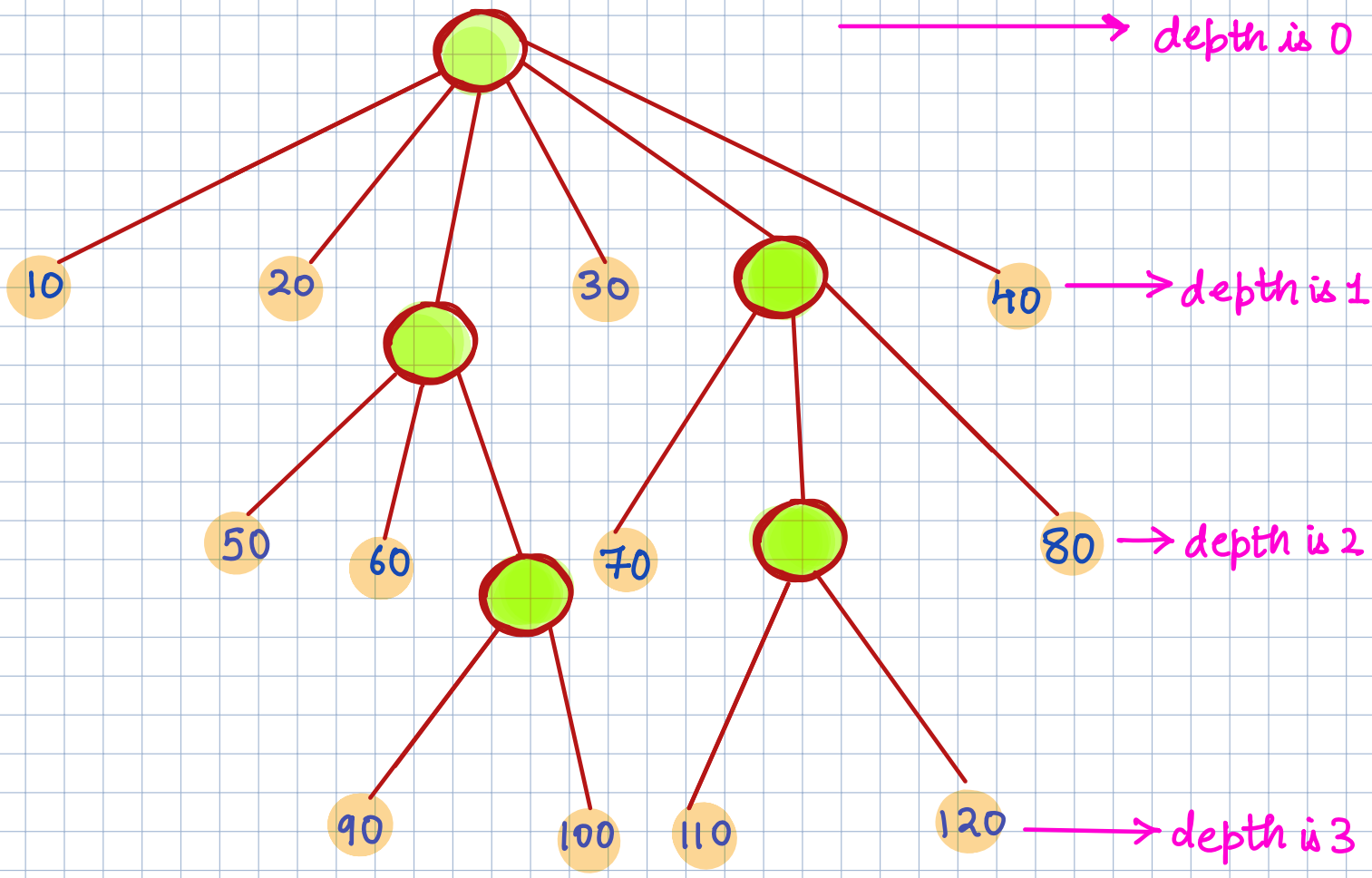
→ [10, 20, [50, 60, [90, 100]], 30, [70, [110, 120], 80], 40]

→ [10, 20, [50, 60, [90, 100]], 30, [70, [110, 120], 80], 40]

# 10, 20 # 50, 60 # 90, 100 $ $ 30 # 70 # 110, 120 $ 80 $ 40 $

Opening square bracket:- #

closing square bracket :- $




arr.flat() ≅ arr.flat(1)

## Array.prototype.flat()

The flat() method creates a new array with all sub-array elements concatenated into it recursively up to the specified depth.

### Syntax

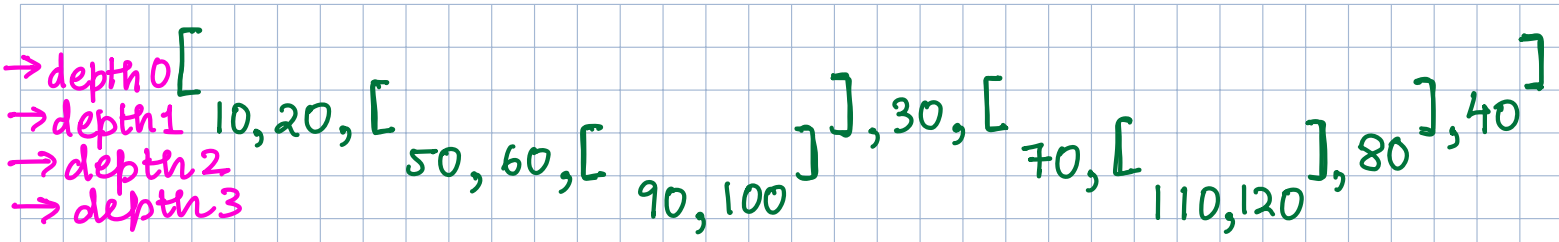
flat()

flat(depth)

### Parameters

depth — Optional
The depth level specifying how deep a nested array structure should be flattened. Defaults to 1.

**Return value** → A new array with the sub-array elements concatenated into it.

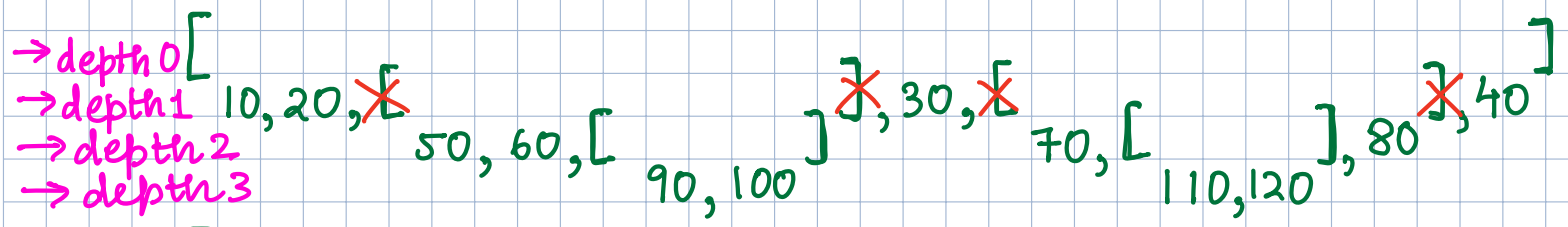


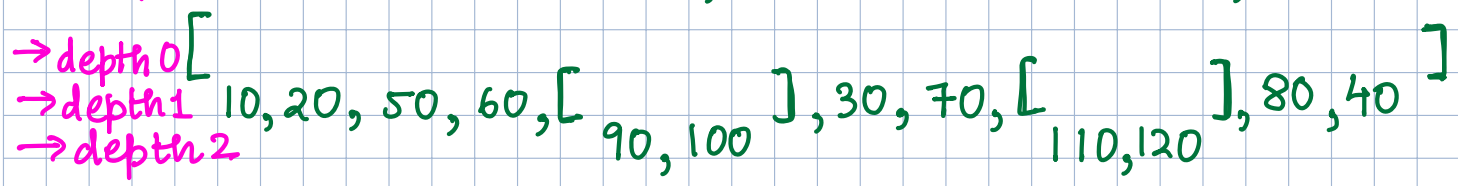

let res 1 = arr. flat(1)

↳ burst the opening and closing bracket of depth 1.

elements of depth 1 ↑ elements of depth 2

elements of depth 2 ↑ elements of depth 3



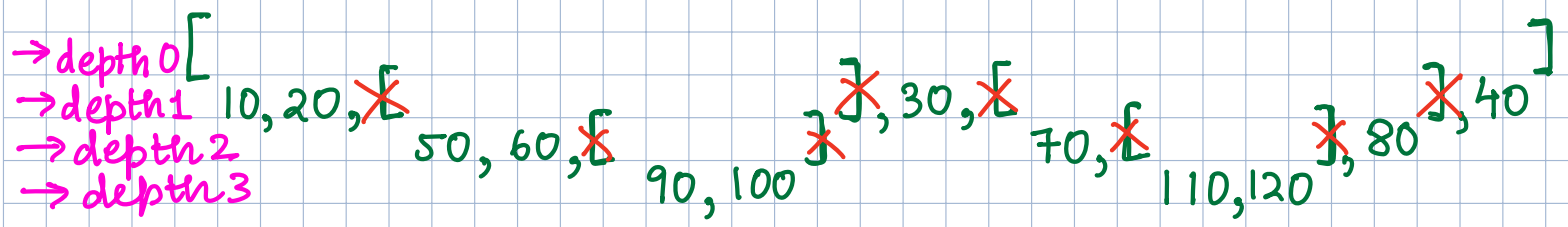



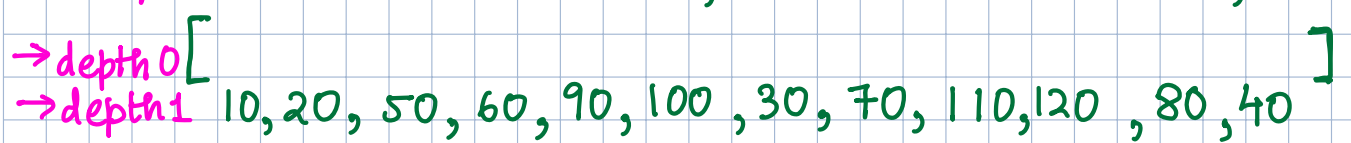

let res2 = arr. flat(2)

↳ burst the opening and closing bracket of depth 1 and depth 2

elements of depth 1 ↑ elements of depth 2

elements of depth 1 ↑ elements of depth 3



```js
let arr = [10, 20, [50, 60, [90, 100]], 30, [70, [110, 120], 80], 40];

let res1 = arr.flat();  // [ 10, 20, 50, 60, [ 90, 100 ], 30, 70, [ 110, 120 ], 80, 40 ]
console.log(res1);

let res2 = arr.flat(1); // [ 10, 20, 50, 60, [ 90, 100 ], 30, 70, [ 110, 120 ], 80, 40 ]
console.log(res2);

let res3 = arr.flat(2); // [10, 20, 50,  60,  90, 100, 30, 70, 110, 120, 80, 40]
console.log(res3);
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
→  Lecture_39 git:(main) ✗ node 1_ArrayFlatDemo.js
[ 10, 20, 50, 60, [ 90, 100 ], 30, 70, [ 110, 120 ], 80, 40 ]
[ 10, 20, 50, 60, [ 90, 100 ], 30, 70, [ 110, 120 ], 80, 40 ]
[
   10, 20, 50,  60,  90,
  100, 30, 70, 110, 120,
   80, 40
]
→  Lecture_39 git:(main) ✗
```

```js
let arr2 = [10, 20, [30, [50, [70, 80, 90], 60], 40], 100, [120, [150, [170, 180, 190], 160], 140, 200], 110];

let res4 = arr2.flat(0);
console.log(res4);
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
→  Lecture_39 git:(main) ✗ node 1_ArrayFlatDemo.js
[
  10,
  20,
  [ 30, [ 50, [Array], 60 ], 40 ],
  100,
  [ 120, [ 150, [Array], 160 ], 140, 200 ],
  110
]
→  Lecture_39 git:(main) ✗
```

```js
let arr2 = [10, 20, [30, [50, [70, 80, 90], 60], 40], 100, [120, [150, [170, 180, 190], 160], 140, 200], 110];

// let res4 = arr2.flat(0);
// console.log(res4);

let res5 = arr2.flat(1);
console.log(res5);
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
→  Lecture_39 git:(main) ✗ node 1_ArrayFlatDemo.js
[
  10,
  20,
  30,
  [ 50, [ 70, 80, 90 ], 60 ],
  40,
  100,
  120,
  [ 150, [ 170, 180, 190 ], 160 ],
  140,
  200,
  110
]
→  Lecture_39 git:(main) ✗
```

```
16  let arr2 = [10, 20, [30, [50, [70, 80, 90], 60], 40], 100, [120, [150, [170, 180, 190], 160], 140, 200], 110];
17
18  // let res4 = arr2.flat(0);
19  // console.log(res4);
20
21  // let res5 = arr2.flat(1);
22  // console.log(res5);
23
24  let res6 = arr2.flat(2);
25  console.log(res6);
26
```

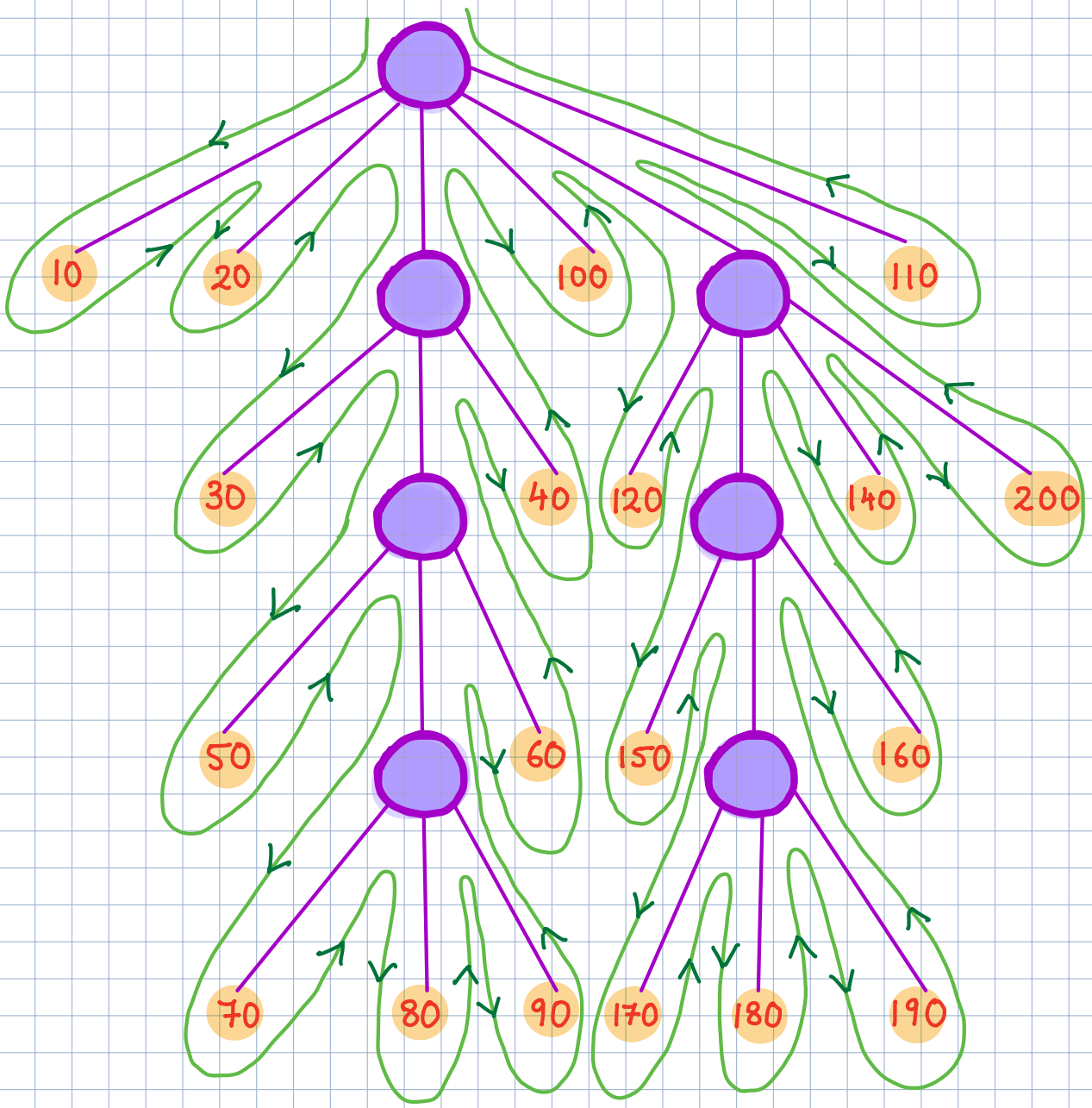
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
→  Lecture_39 git:(main) ✗ node 1_ArrayFlatDemo.js
[
  10,
  20,
  30,
  50,
  [ 70, 80, 90 ],
  60,
  40,
  100,
  120,
  150,
  [ 170, 180, 190 ],
  160,
  140,
  200,
  110
]
→  Lecture_39 git:(main) ✗
```

```
16  let arr2 = [10, 20, [30, [50, [70, 80, 90], 60], 40], 100, [120, [150, [170, 180, 190], 160], 140, 200], 110];
17
18  // let res4 = arr2.flat(0);
19  // console.log(res4);
20
21  // let res5 = arr2.flat(1);
22  // console.log(res5);
23
24  // let res6 = arr2.flat(2);
25  // console.log(res6);
26
27  let res7 = arr2.flat(3);
28  console.log(res7);
29
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
→  Lecture_39 git:(main) ✗ node 1_ArrayFlatDemo.js
[
   10,  20,  30,  50,  70,  80,
   90,  60,  40, 100, 120, 150,
  170, 180, 190, 160, 140, 200,
  110
]
→  Lecture_39 git:(main) ✗
```




[10,20,[30,[50,[70,80,90],60],40],100,[120,[150,[170,180,190],160],140,200],110]

[10,20,[30,[50,[70,80,90],60],40],100,[120,[150,[170,180,190],160],140,200],110]

0 1 2 3 3 2 1 1 2 3 3 2 1 0

depth 0 [ ]
depth 1 10,20,[ ],100,[ ],110
depth 2 30,[ ],40 120,[ ],140,200
depth 3 50,[ ],60 150,[ ],160
depth 4 70,80,90 170,180,190

# CUSTOM FLAT

```
1
2   Array.prototype.myFlat = function(td){
3
4       let oarr = this;
5       let res = [];
6
7       customFlat(oarr, td, res);
8
9       return res;
10  }
11
12  function customFlat(node, td, res){
13      if(Array.isArray(node)){
14      if(td > 0){
15          for(let i = 0; i < node.length; i++){
16              customFlat(node[i], td - 1, res);
17          }
18      }else{
19          for(let i = 0; i < node.length; i++){
20              res.push(node[i]);
21          }
22      }
23      } else {
24          res.push(node);
25      }
26  }
```

```
let arr2 = [10, 20, [30, [50, [70, 80, 90], 60], 40], 100, [120, [150, [170, 180, 190], 160], 140, 200], 110];

let res4 = arr2.myFlat(0);
console.log(res4);

let res5 = arr2.myFlat(1);
console.log(res5);

let res6 = arr2.myFlat(2);
console.log(res6);

let res7 = arr2.myFlat(3);
console.log(res7);

let res8 = arr2.myFlat(Infinity);
console.log(res8);
```

```
→  Lecture_39 git:(main) ✗ node 2_ArrayCustomFlat.js
[
  10,
  20,
  [ 30, [ 50, [Array], 60 ], 40 ],
  100,
  [ 120, [ 150, [Array], 160 ], 140, 200 ],
  110
]
[
  10,
  20,
  30,
  [ 50, [ 70, 80, 90 ], 60 ],
  40,
  100,
  120,
  [ 150, [ 170, 180, 190 ], 160 ],
  140,
  200,
  110
]
[
  10,
  20,
  30,
  50,
  [ 70, 80, 90 ],
  60,
  40,
  100,
  120,
  150,
  [ 170, 180, 190 ],
  160,
  140,
  200,
  110
]
[
   10,  20,  30,  50,  70,  80,
   90,  60,  40, 100, 120, 150,
  170, 180, 190, 160, 140, 200,
  110
]
[
   10,  20,  30,  50,  70,  80,
   90,  60,  40, 100, 120, 150,
  170, 180, 190, 160, 140, 200,
  110
]
```

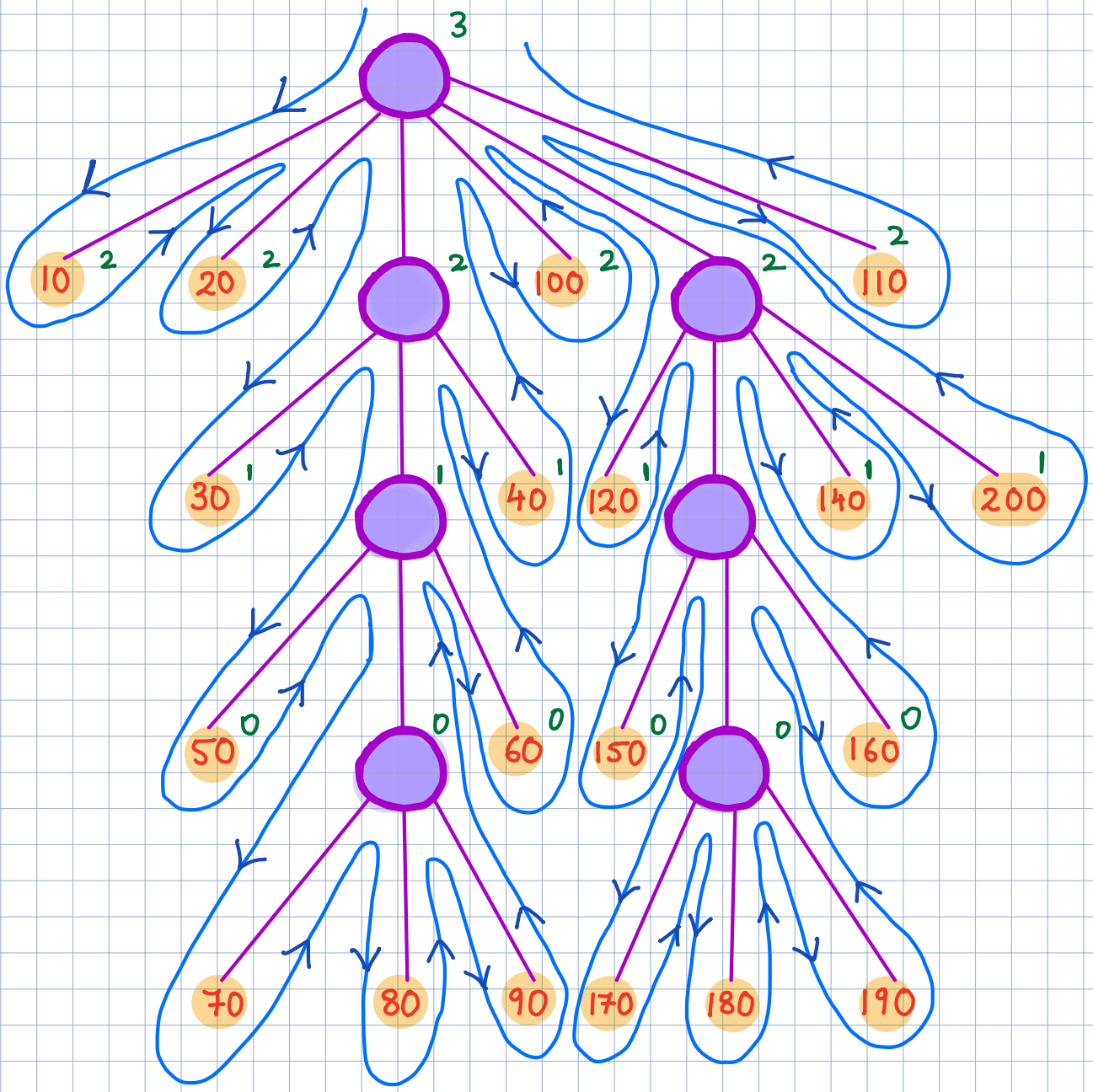
arr2.myFlat(0)

arr2.myFlat(1)

arr2.myFlat(2)

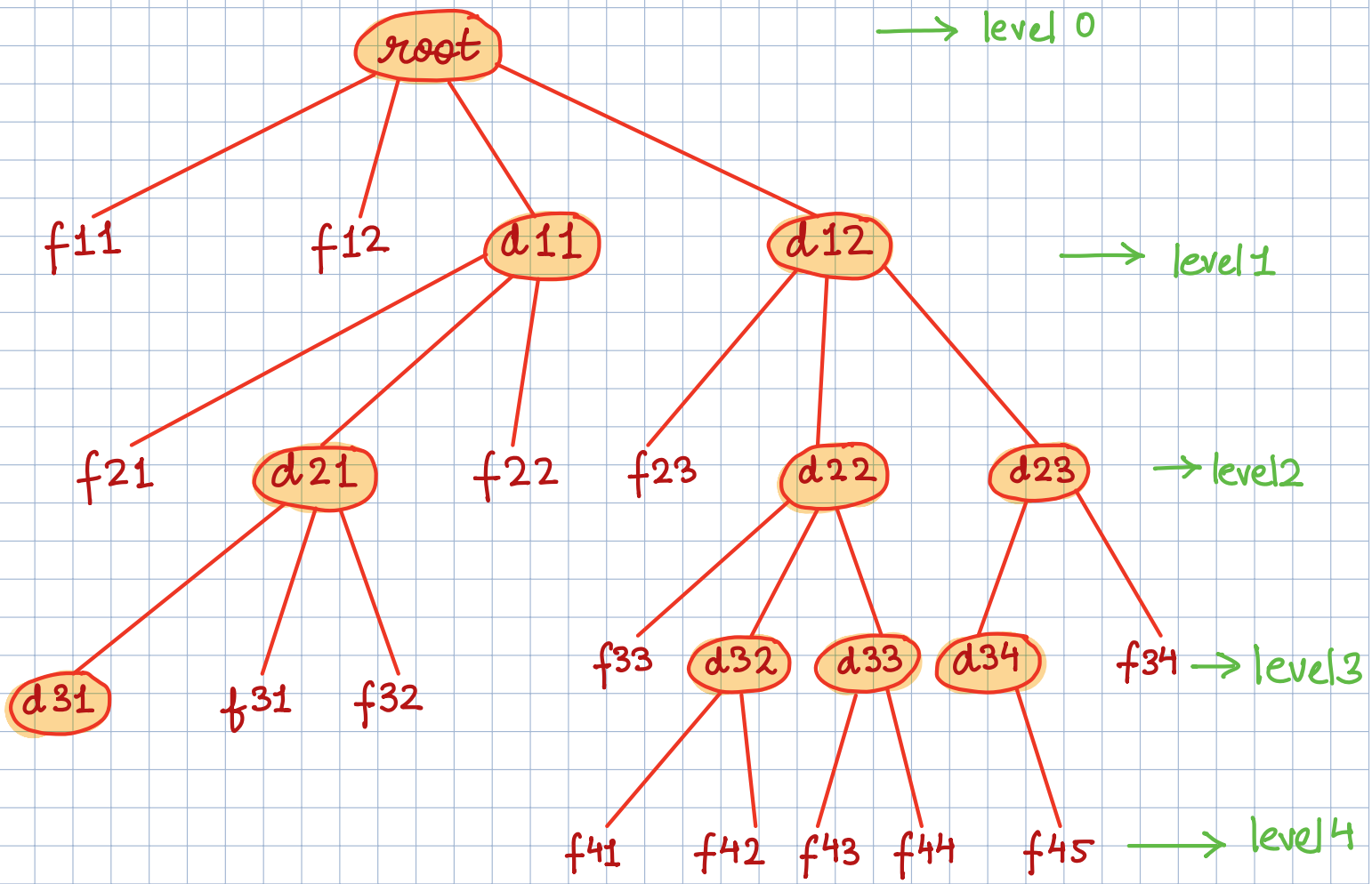
arr2.myFlat(3)

arr2.myFlat(Infinity)

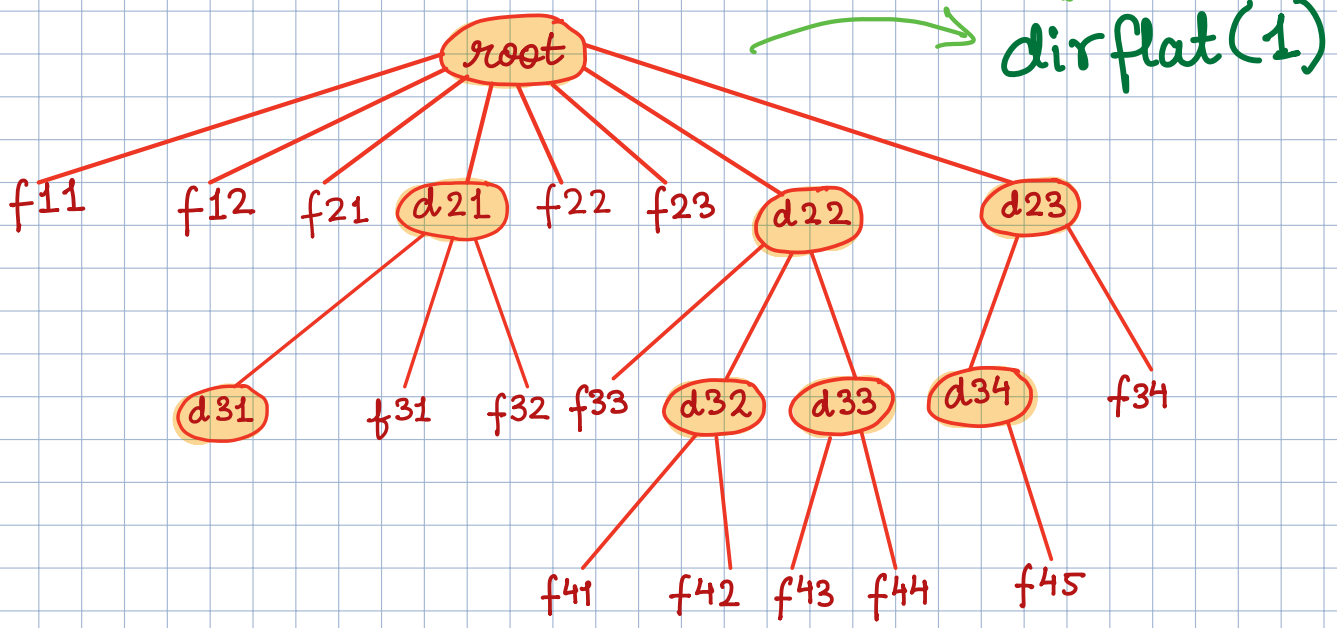

3
10
2
20
2
2
100
2
2
110
2
30
1
1
40
1
120
1
140
1
200
1
50
0
0
60
0
150
0
0
160
0
70
80
90
170
180
190

Ques :- folder directories

we now need to do dirflat(1)




Ans for dirflat(1)



# ARRAY FLATMAP

## Array.prototype.flatMap()

The flatMap() method returns a new array formed by applying a given a given callback function to each element of the array, and then flattening the result by one level. It is identical to a map() followed by a flat() of depth 1, but slightly more efficient than calling those two methods separately.

# ARRAY CONSTRUCTION

## Array.of()

The Array.of() method creates a new Array instance from a variable number of arguments, regardless of number or type of the arguments.

## Array.from()

The Array.from() static method creates a new, shallow-copied Array instance from an array-like or iterable object.

```
let arr1 = [10, 20, 30];
console.log(arr1);

let arr2 = Array.of(10);
console.log(arr2);

let arr3 = Array.of(10, 20, 30, 40, 50);
console.log(arr3);

let arr4 = Array.from("Hemakshi"); // array like objects (for instance string, nodelist, arguments)
console.log(arr4);

let arr5 = arr4.map(ch => ch. charCodeAt(0)+ 1);
// let arr5 = arr4.map(ch => String.fromCharCode(ch. charCodeAt(0)+ 1));
console.log(arr5);

let arr6 = arr5.map(v => String.fromCharCode(v))
console.log(arr6);

let arr7 = arr4.map((ch, i) => i <= 1? String.fromCharCode(ch.charCodeAt(0) + 1) : String.fromCharCode(ch.charCodeAt(0) - 1));
console.log(arr7);

let str = arr6.join("");
console.log(str);
```

```
→  Lecture_39 git:(main) ✗ node 3_ArrayConstruction.js
[ 10, 20, 30 ]
[ 10 ]
[ 10, 20, 30, 40, 50 ]
[
  'H', 'e', 'm',
  'a', 'k', 's',
  'h', 'i'
]
[
   73, 102, 110,  98,
  108, 116, 105, 106
]
[
  'I', 'f', 'n',
  'b', 'l', 't',
  'i', 'j'
]
[
  'I', 'f', 'l',
  '`', 'j', 'r',
  'g', 'h'
]
Ifnbltij
→  Lecture_39 git:(main) ✗
```

# ARRAY BULK UPDATE

## Array.prototype.fill()

The fill() method changes all elements in an array to a static value, from a start index (default 0) to an end index (default array.length). It returns the modified array.

### Syntax

```
fill(value)
fill(value, start)
fill(value, start, end)
```

- value → value to fill the array with.
- start → (optional) Start index (inclusive), default 0.
- end → (optional) End index (exclusive), default arr.length.

### Return value

The modified array filled with value.

# Array.prototype.copyWithin()

The copyWithin() method shallow copies part of an array to another location in the same array and returns it without modifying its length.

## Syntax

```
copyWithin(target)
copyWithin(target, start)
copyWithin(target, start, end)
```

```
1
2   let arr = [10, 20, 30, 40, 50, 60];
3   arr.fill(5, 1, 4);
4   console.log(arr);
5
6   arr.fill(7, 2);
7   console.log(arr);
8
9   arr.fill(8);
10  console.log(arr);
11
12  let arr1 = [10, 20, 30, 40, 50, 60];
13  arr1.copyWithin(2, 4, 6);
14  console.log(arr1);
15
16  let arr2 = [10, 20, 30, 40, 50, 60, 70, 80, 90];
17  arr2.copyWithin(3, 0, 2);
18  console.log(arr2);
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

→  Lecture_39 git:(main) ✗ node 4_ArrayBulkUpdate.js
[ 10, 5, 5, 5, 50, 60 ]
[ 10, 5, 7, 7, 7, 7 ]
[ 8, 8, 8, 8, 8, 8 ]
[ 10, 20, 50, 60, 50, 60 ]
[
  10, 20, 30, 10, 20,
  60, 70, 80, 90
]
→  Lecture_39 git:(main) ✗
```