Teknik Informatika | Kelas B

Stack & Tree

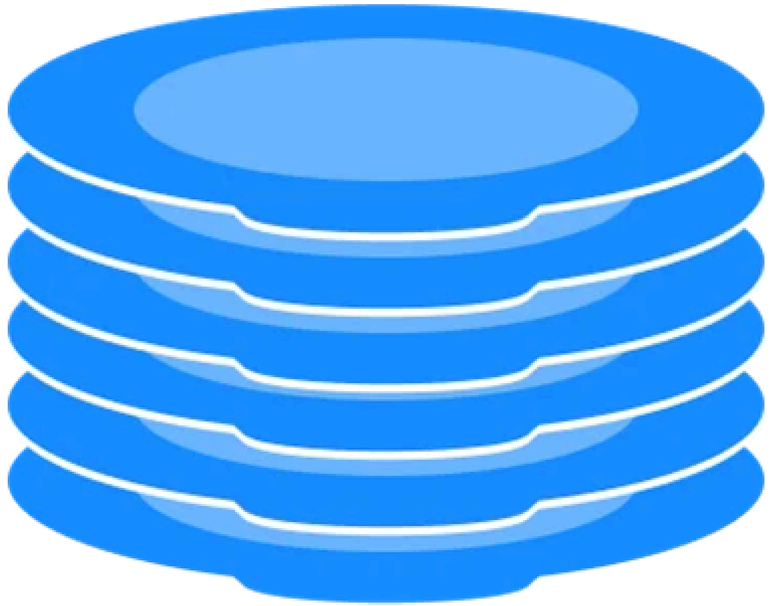
Struktur Data

Ady Ulil Amri (D121231080)
Ramadani (D121231014)
Muhammad Fauzan Rusda (D121231035)
Taufiqurrahman Hendra (D121231071)
Muhammad Faaiq Fadhlurrahman (D121231101)

Stack

Tumpukan adalah struktur data linier yang mengikuti prinsip Last In First Out (LIFO). Ini berarti elemen terakhir yang dimasukkan ke dalam tumpukan akan dihapus terlebih dahulu.

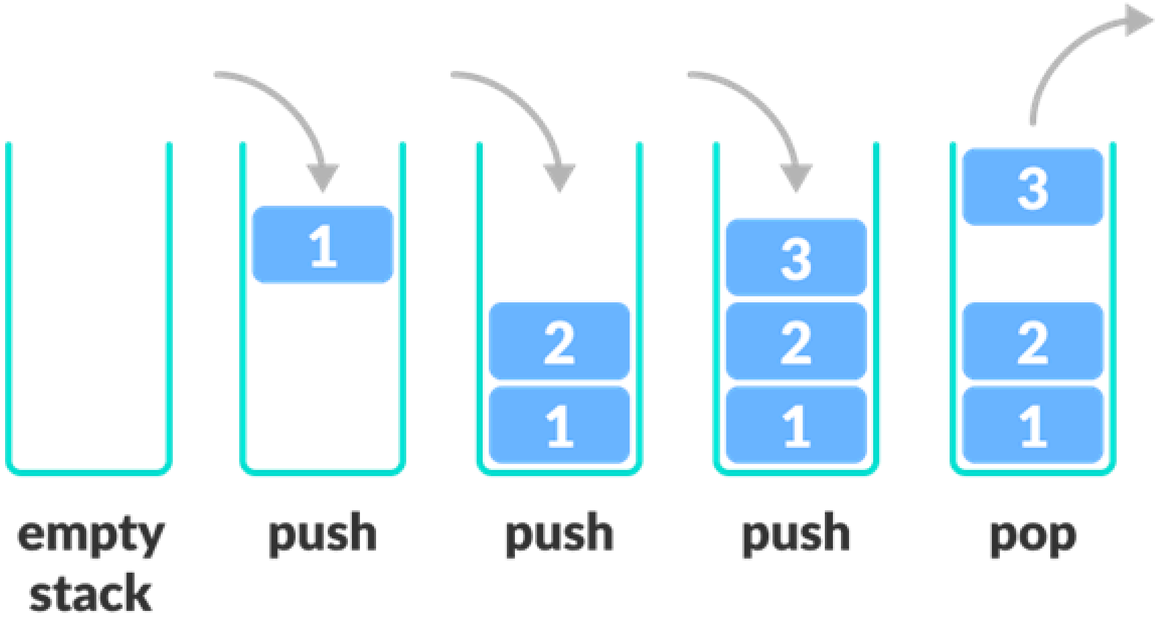
Anda bisa membayangkan struktur data stack sebagai tumpukan piring di atas piring lainnya.



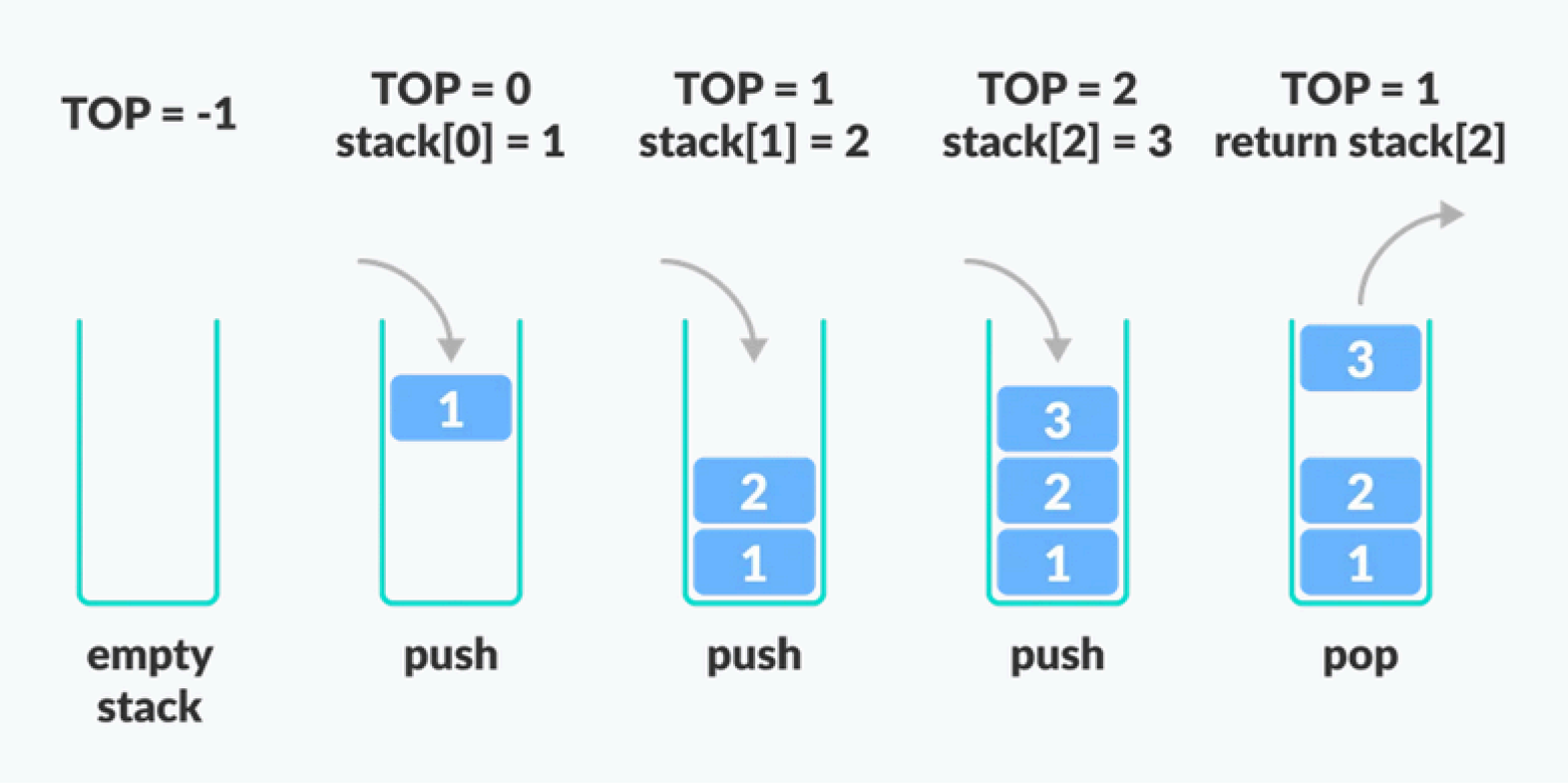
Operasi Dasar Stack

Ada beberapa operasi dasar yang memungkinkan kita untuk melakukan berbagai tindakan pada stack.

- **Push**
- **Pop**
- **IsEmpty**
- **IsFull**
- **Peek**



Cara Kerja Stack



Operasi Stack

Push

```
#Menerapkan Push
print("Menerapkan Push")
stack = []
print(stack)

stack.append('1')
print(stack)
```

```
Menerapkan Push
[]
['1']
```

Pop

```
#Menerapkan Pop
print("\nMenerapkan Pop")
stack = [1, 2, 3, 4, 5]
print(stack)

stack.pop()
print(stack)
```

```
Menerapkan Pop
[1, 2, 3, 4, 5]
[1, 2, 3, 4]
```

Peek

```
#Menerapkan Peek
print("\nMenrapkan Peek")
stack = [1, 2, 3, 4, 5]
def peek(stack):
    return stack[-1]

print(peek(stack))
```

```
Menrapkan Peek
5
```

Operasi Stack

IsEmpty

```
#Menerapkan isEmpty
print("\nMenerapkan isEmpty")

stack = []
def isEmpty(stack):
    return len(stack) == 0
print(isEmpty(stack))
```

```
Menerapkan isEmpty
True
```

IsFull

```
#Menerapkan isFull
print("\nMenerapkan isFull")

stack = []
capacity = 3
def isFull(stack):
    return len(stack) == capacity

stack.append(1)
stack.append(2)
stack.append(3)
stack.pop()

if(isFull(stack)):
    print('stack is full')
else :
    print('stack not full')
```

```
Menerapkan isFull
stack not full
```

Real Case Stack

Anda sedang mencatat skor untuk permainan bisbol dengan aturan yang unik. Permainan terdiri dari beberapa putaran, di mana skor dari putaran sebelumnya dapat memengaruhi skor putaran berikutnya.

Pada awal permainan, Anda memulai dengan catatan kosong. Anda diberikan daftar string ops, di mana ops[i] adalah operasi ke-i th yang harus Anda terapkan pada catatan dan salah satunya adalah sebagai berikut:

1. Sebuah bilangan bulat x - catat skor baru x.
2. "+" Catat skor baru yang merupakan jumlah dari dua skor sebelumnya.
3. "D" Catat skor baru yang dua kali lipat dari skor sebelumnya. .
4. "C" Batalkan skor sebelumnya, menghapusnya dari catatan.

Note : Untuk operasi kedua, tiga dan empat. Jika skor sebelumnya tidak ada kembalikan nilai -1

Kembalikan jumlah semua skor pada catatan.

Contoh 1:
Input : ops = ["5", "2", "C", "D", "+"]
Output : 30

Contoh 2:
Input : ops = ["5", “-2", “4", “C", “D“, “9”, “+”, “+”]
Output : 27

Real Case Stack

Contoh 1 :

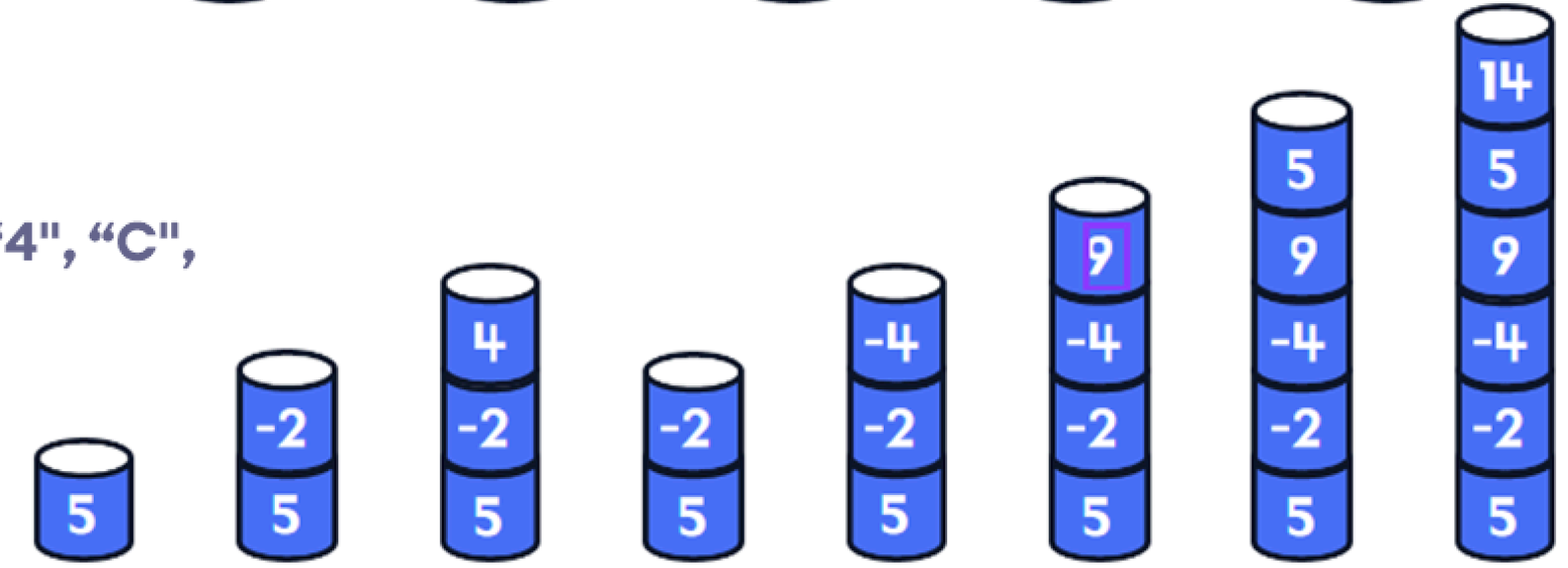
Input : ops = ["5", "2", "C", "D", "+"]

Output = 30

Contoh 2 :

Input : ops = ["5", “-2", “4", “C", “D“, “9”, “+”, “+”]

Output = 27



Real Case Stack : Baseball Game

pas.py > ...

```python
def calculate_score(ops):
    # Inisialisasi stack untuk menyimpan skor yang valid
    stack = []
```

Real Case Stack : Baseball Game

```
# Iterasi melalui setiap operasi dalam daftar ops
for op in ops:
    if op.isdigit() or (op[0] == '-' and op[1:].isdigit()): # Jika op adalah bilangan positif atau negatif, tambahkan sebagai skor baru
        stack.append(int(op))
    elif op == "+":                             # Jika op adalah "+", jumlahkan dua skor terakhir
        if len(stack) >= 2:                     # Periksa apakah ada dua skor sebelumnya
          stack.append(stack[-1] + stack[-2])   # Tambahkan hasil penjumlahan dalam stack
        else:
            # Tangani operasi "+" yang tidak valid (tidak ada skor sebelumnya)
            return -1                           # Atau munculkan exception
    elif op == "D":                             # Jika op adalah "D", gandakan skor terakhir
        if len(stack) >= 1:                     # Periksa apakah ada satu skor sbelumnya
            stack.append(stack[-1] * 2)         # Tambahka skor ganda ke stack
        else:
            # Tangani operasi "D" yang tidak valid (tidak ada skor sebelumnya)
            return -1
    elif op == "C":                             # Jika op adalah "C", hapus skor terakhir
        if len(stack) >= 1:                     # Periksa apakah ada satu skor sbelumnya
            stack.pop()                         # Hapus skor terakhir dari stack
        else:
            # Tangani operasi "C" yang tidak valid (tidak ada skor sebelumnya)
            return -1

# Kembalikan jumlah semua skor di dalam stack
return sum(stack)
```

Real Case Stack : Baseball Game

```
# Contoh Penggunaan
print("Example 1")
ops = ["5", "2", "C", "D", "+"]
print(f"Input  : {ops}")
print(f"Output : {calculate_score(ops)}")

# Contoh penggunaan lain
print("\nExample 2")
ops = ["5","-2","4","C","D","9","+","+"]
print(f"Input  : {ops}")
print(f"Output : {calculate_score(ops)}")
```

Real Case Stack : Baseball Game

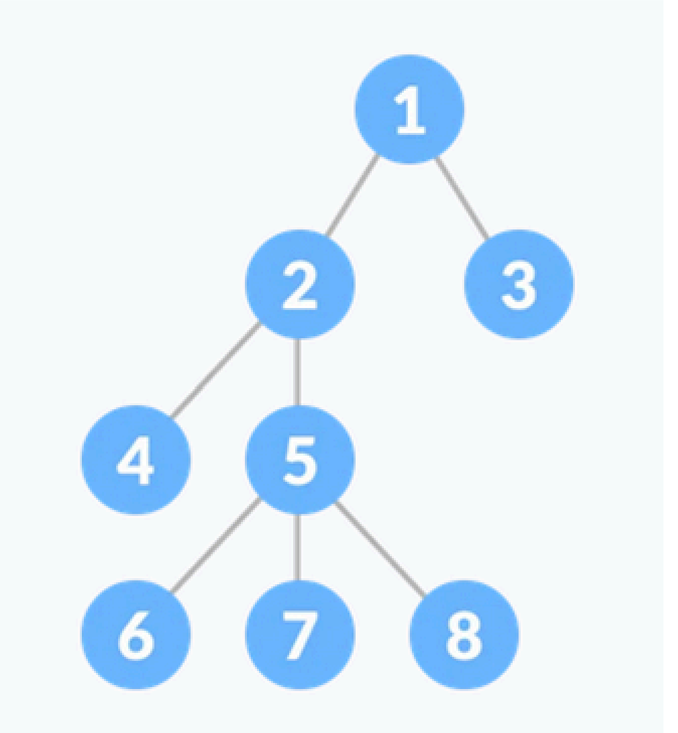
Capture Output :

```
PS C:\Users\M S I\OneDrive\Documents\PKM-KC> python -u "c:\Use
Example 1
Input  : ['5', '2', 'C', 'D', '+']
Output : 30

Example 2
Input  : ['5', '-2', '4', 'C', 'D', '9', '+', '+']
Output : 27
```

Apa itu Tree?

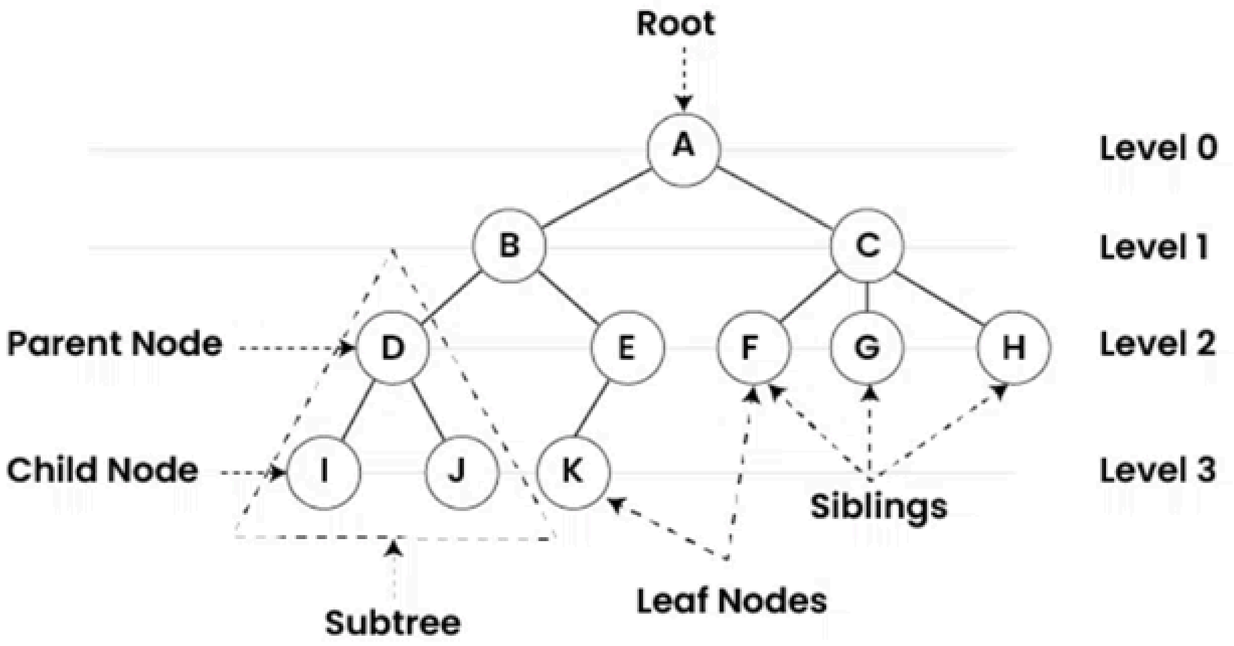
Tree adalah Struktur data non-linear di mana kumpulan elemen (nodes) terhubung satu sama lain melalui sisi sedemikian rupa sehingga ada tepat satu jalur antara dua node.



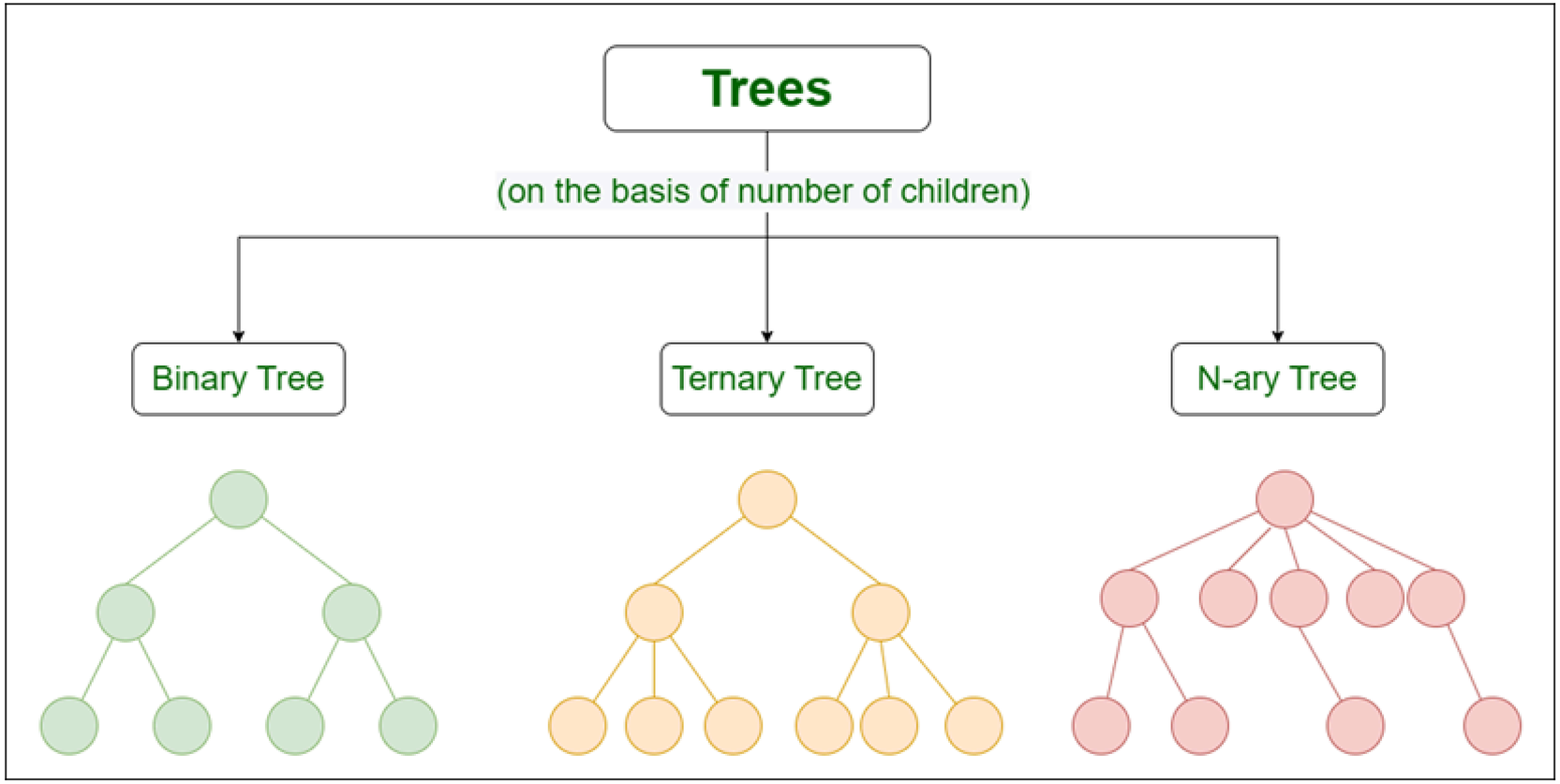
Terminologi Tree

Tree

Data Structure



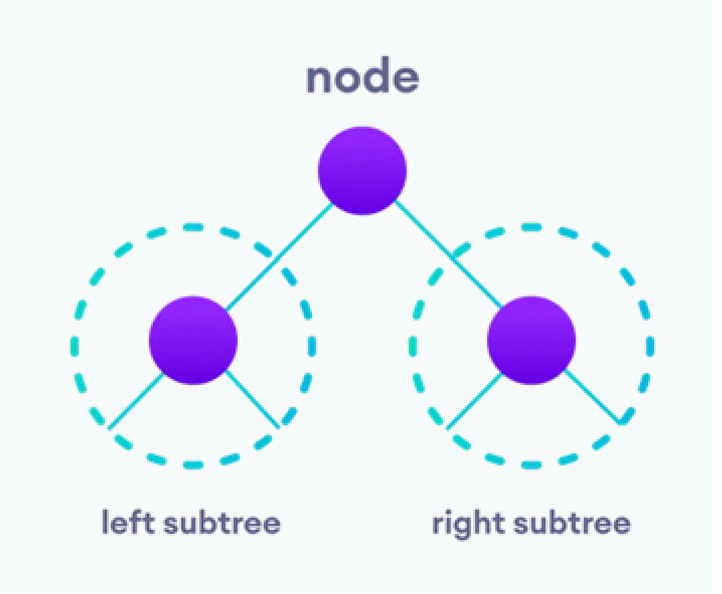
Tipe Tree


Trees
(on the basis of number of children)
Binary Tree
Ternary Tree
N-ary Tree

Tree Traversal

Struktur data linear seperti array, stack, queues, dan linked list hanya memiliki satu cara untuk membaca data. Namun struktur data hirarkis seperti pohon dapat dilalui dengan berbagai cara, yakni :

- **Inorder Traversal**
- **Preorder Traversal**
- **Postorder Traversal**



Real Case Tree : Coin Change

Diberikan sebuah kumpulan koin dengan berbagai nilai, yang diwakili oleh array integer 'coins' berukuran 'N', dan sebuah nilai yang diinginkan, disimbolkan dengan 'sum'. Tugasnya adalah menemukan berapa banyak cara yang mungkin untuk mencapai jumlah nilai tersebut menggunakan kombinasi koin dari kumpulan 'coins'.
Catatan: Anda dapat menggunakan setiap jenis koin sebanyak yang Anda inginkan, dan asumsikan Anda memiliki pasokan yang tidak terbatas dari setiap jenis koin.

Contoh 1:

Input:

N = 3, sum = 4

coins = {1,2,3}

Output: 4

Penjelasan: Ada empat cara yang mungkin: {1,1,1,1}, {1,1,2}, {2,2}, dan {1,3}.

Contoh 2:

Input:

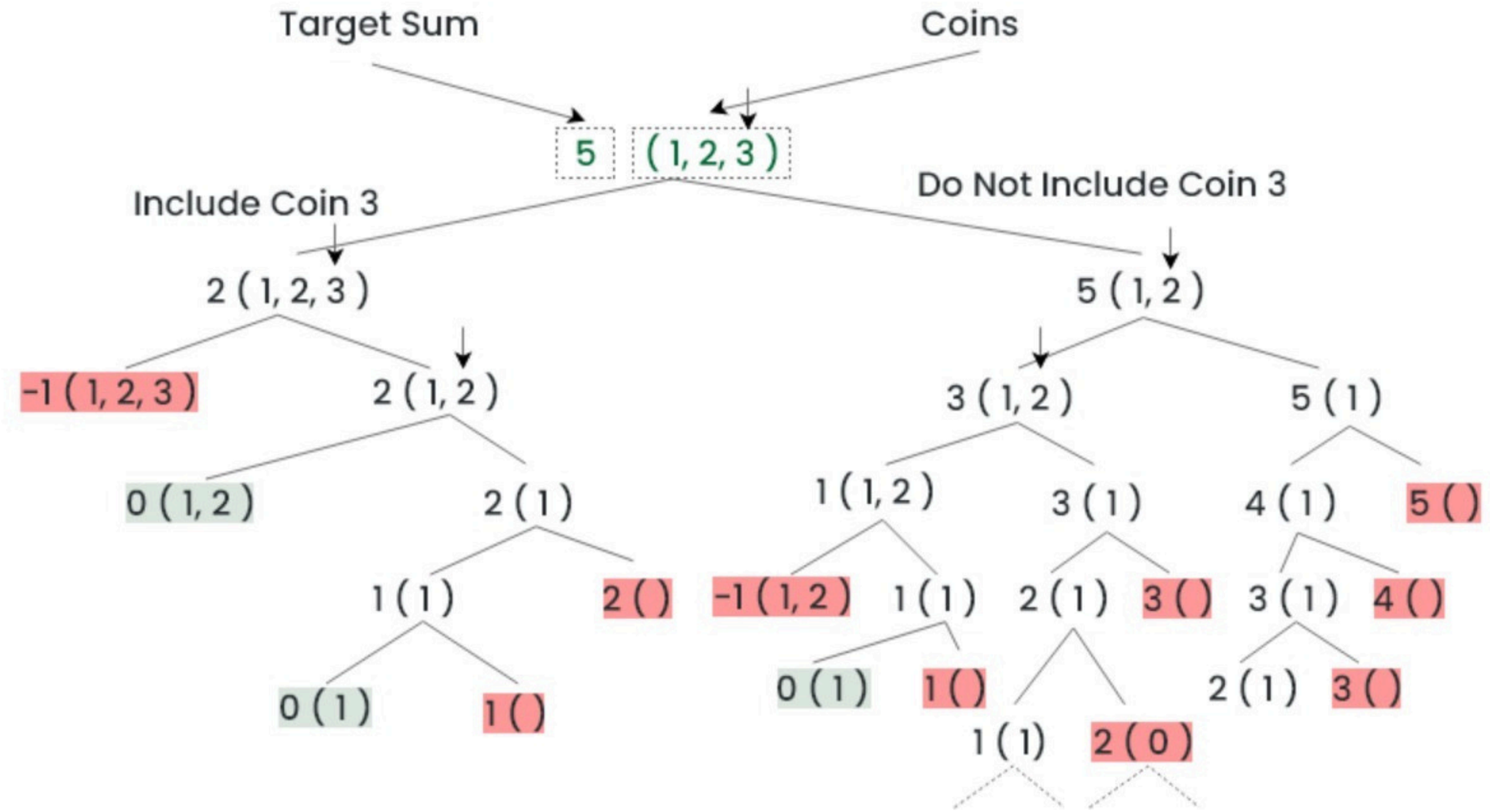
N = 4, Sum = 10

coins = {2,5,3,6}

Output: 5

Penjelasan: Ada lima cara yang mungkin: {2,2,2,2,2}, {2,2,3,3}, {2,2,6}, {2,3,5}, dan {5,5}.

Real Case Tree : Coin Change



Real Case Tree : Coin Change

```c
#include <stdio.h>
#include <stdlib.h>

// Struktur node untuk pohon
struct Node {
    int value; // Nilai koin yg dipertimbangkan
    int remainingSum; // Jumlah sum yang tersisa
    struct Node* left; // Anak node kiri
    struct Node* right; // Anak node kanan
};

// Fungsi untuk membuat node baru
struct Node* createNode(int value, int remainingSum) {
    // Mengalokasikan memori untuk node baru
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    // Mengatur nilai dan sum yang tersisa
    newNode->value = value;
    newNode->remainingSum = remainingSum;
    newNode->left = NULL; // Awalnya tidak memiliki anak kiri
    newNode->right = NULL; // Awalnya tidak memiliki anak kanan
    return newNode;
}
```

Real Case Tree : Coin Change

```c
int countCombinations(int coins[], int n, int remainingSum) {
    // Jika remainingSum menjadi 0, satu kombinasi ditemukan
    if (remainingSum == 0)
        return 1;
    // Jika tidak ada koin tersisa atau remainingSum menjadi negatif, tidak ada kombinasi yang mungkin
    if (n <= 0 || remainingSum < 0)
        return 0;

    // Buat node baru dengan nilai koin terakhir
    struct Node* current = createNode(coins[n - 1], remainingSum);

    // Hitung jumlah kombinasi untuk sisa remainingSum setelah menggunakan koin terakhir
    int include = countCombinations(coins, n, current->remainingSum - current->value);

    // Hitung jumlah kombinasi untuk remainingSum tanpa menggunakan koin terakhir
    int exclude = countCombinations(coins, n - 1, current->remainingSum);

    // Gabungkan jumlah kombinasi dari dua kasus di atas
    return include + exclude;
}
```

Real Case Tree : Coin Change

```c
int main() {
    // Nilai sum dan array koin
    int sum = 4;
    int coins[] = {1, 2, 3};
    int n = sizeof(coins) / sizeof(coins[0]);

    // Menghitung dan mencetak jumlah kombinasi koin
    printf("Jumlah kombinasi koin untuk mencapai nilai %d: %d\n", sum, countCombinations(coins, n, sum));

    return 0;
}
```

```
Jumlah kombinasi koin untuk mencapai nilai 4: 4
```

Real Case Tree : Subordinates

Time limit: 1.00 s **Memory limit:** 512 MB

Given the structure of a company, your task is to calculate for each employee the number of their subordinates.

Input

The first input line has an integer n: the number of employees. The employees are numbered $1, 2, \ldots, n$, and employee 1 is the general director of the company.

After this, there are $n-1$ integers: for each employee $2, 3, \ldots, n$ their direct boss in the company.

Output

Print n integers: for each employee $1, 2, \ldots, n$ the number of their subordinates.

Real Case Tree : Subordinates

Constraints

- $1 \leq n \leq 2 \cdot 10^5$

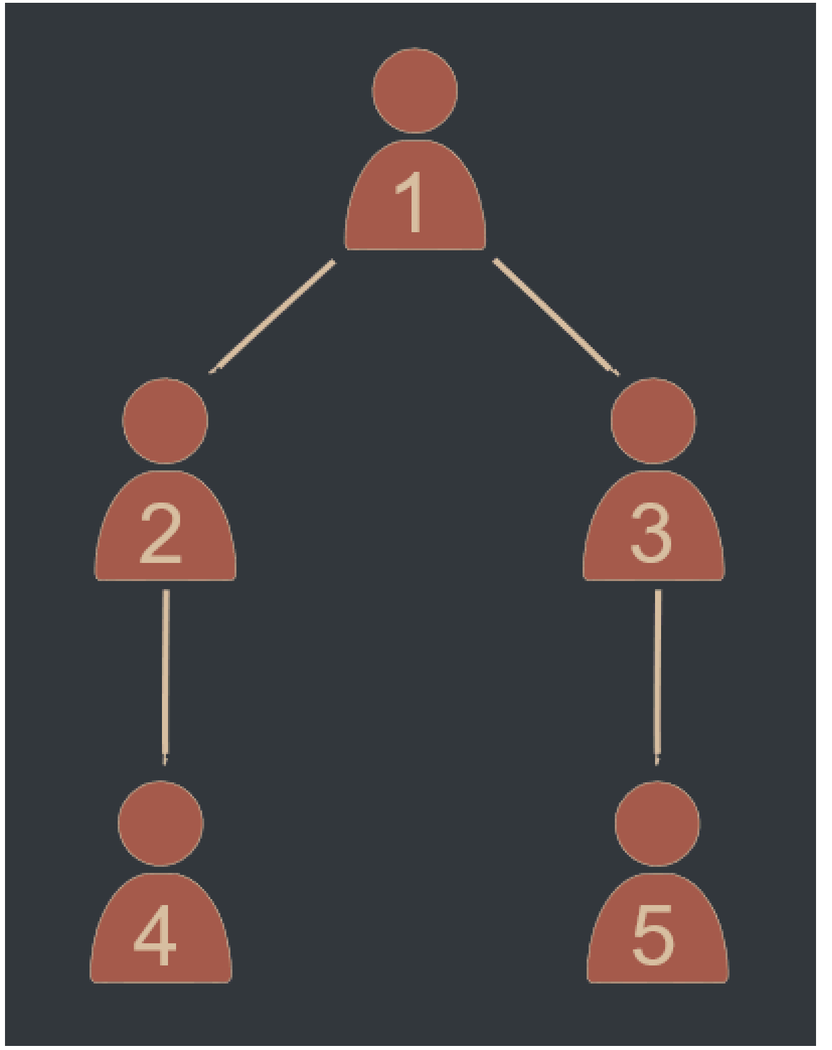
Example

Input:

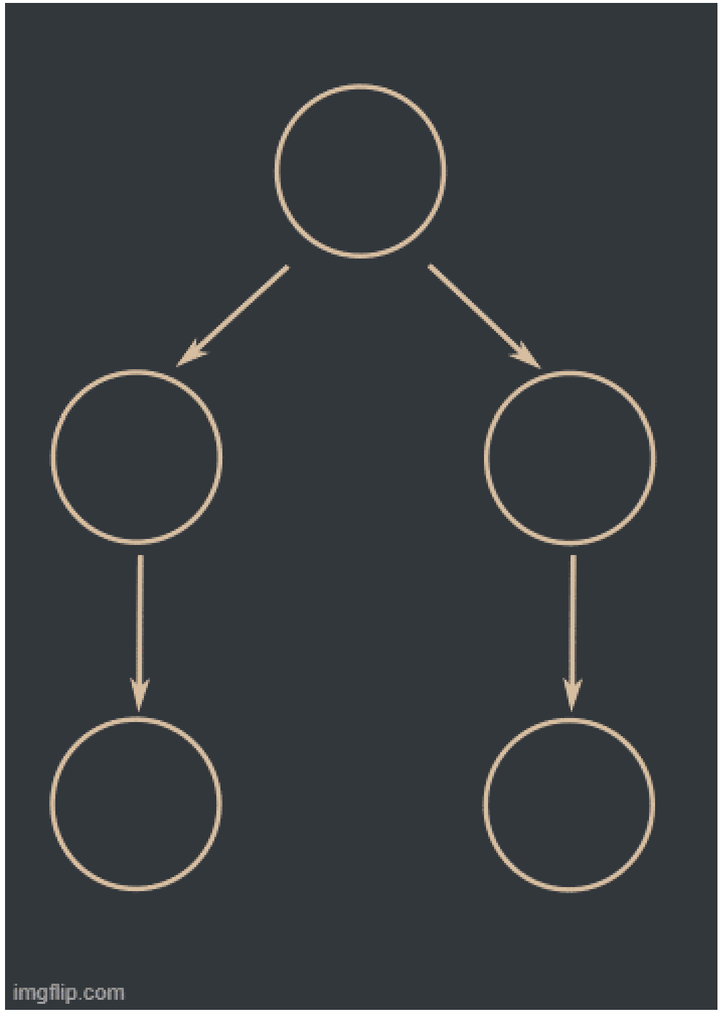
```
5
1 1 2 3
```

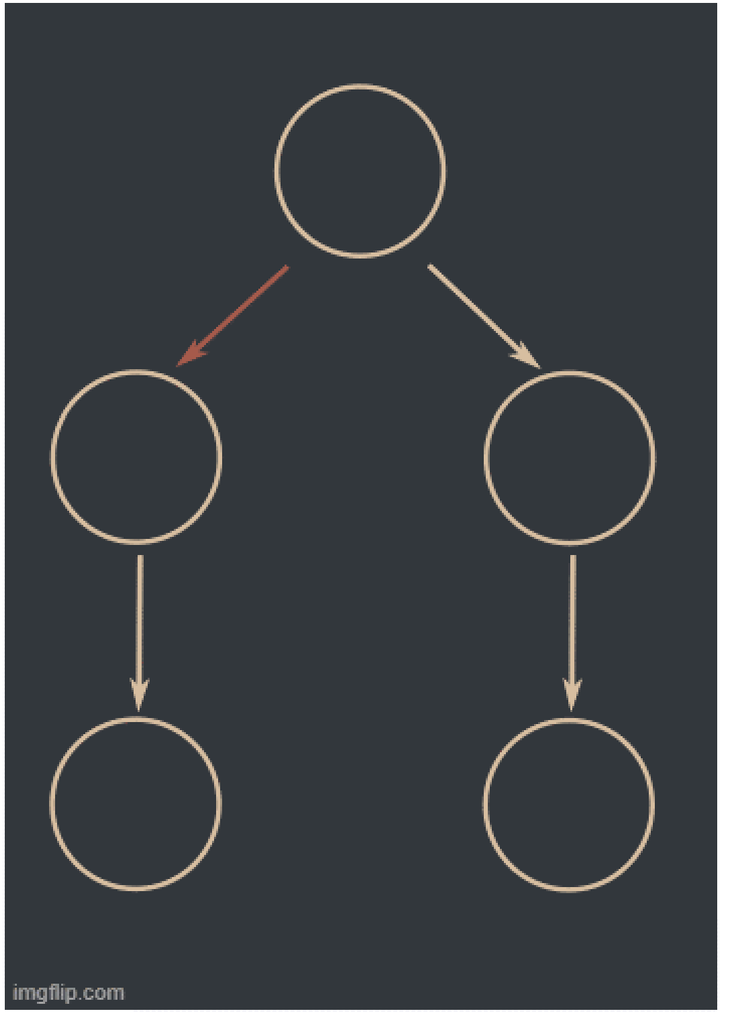
Output:

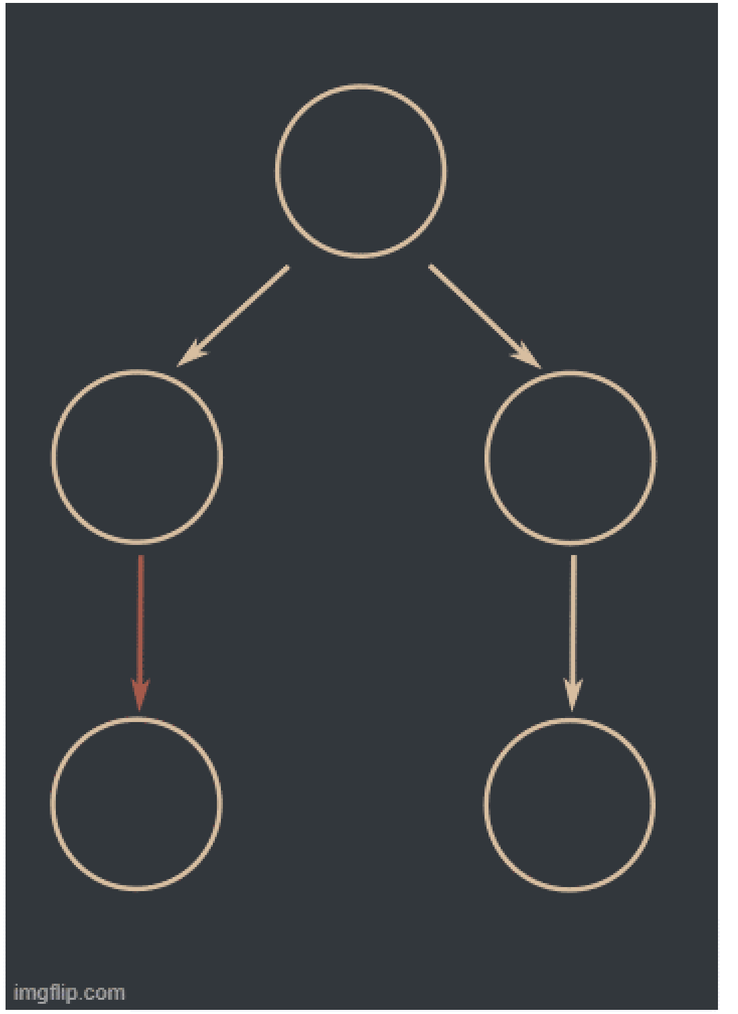
```
4 1 1 0 0
```



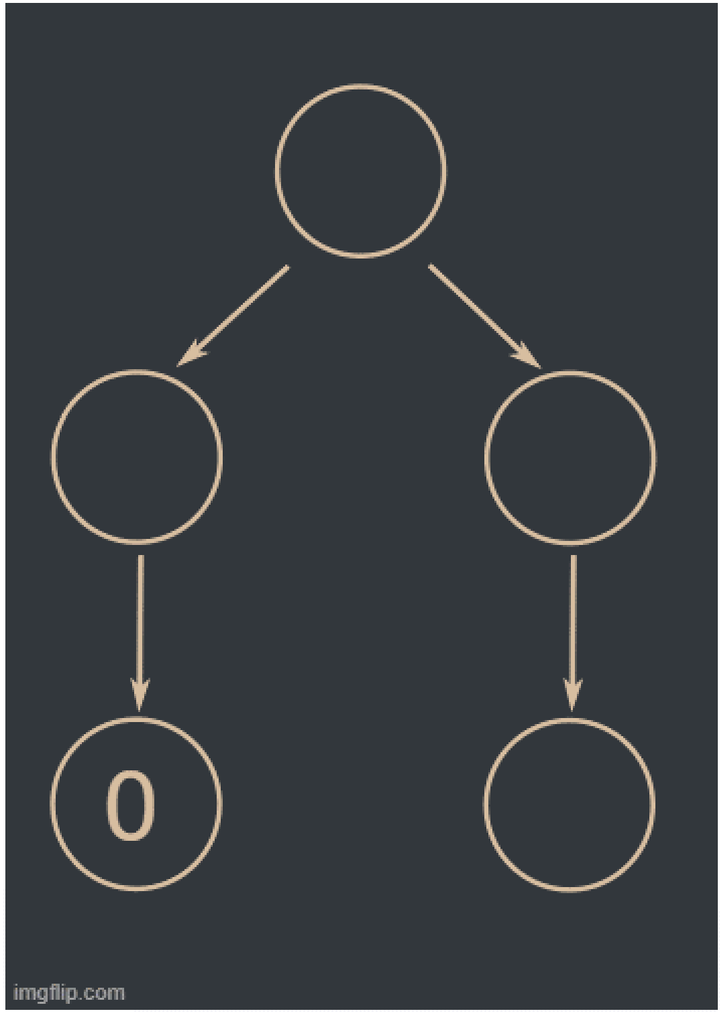
Real Case Tree : Subordinates

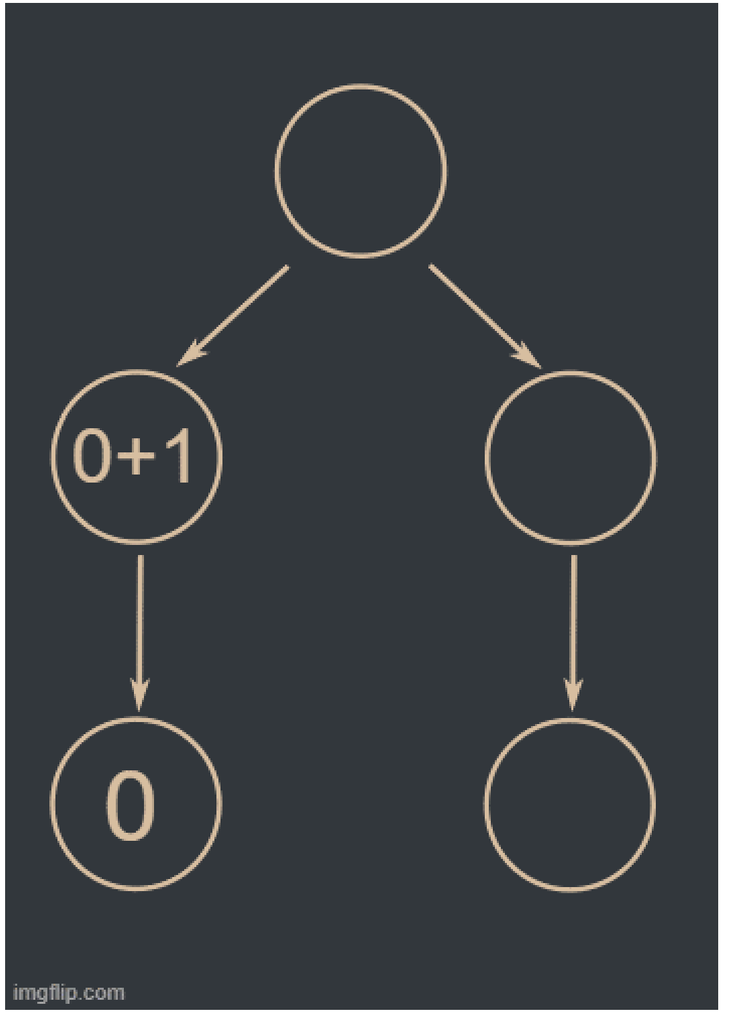


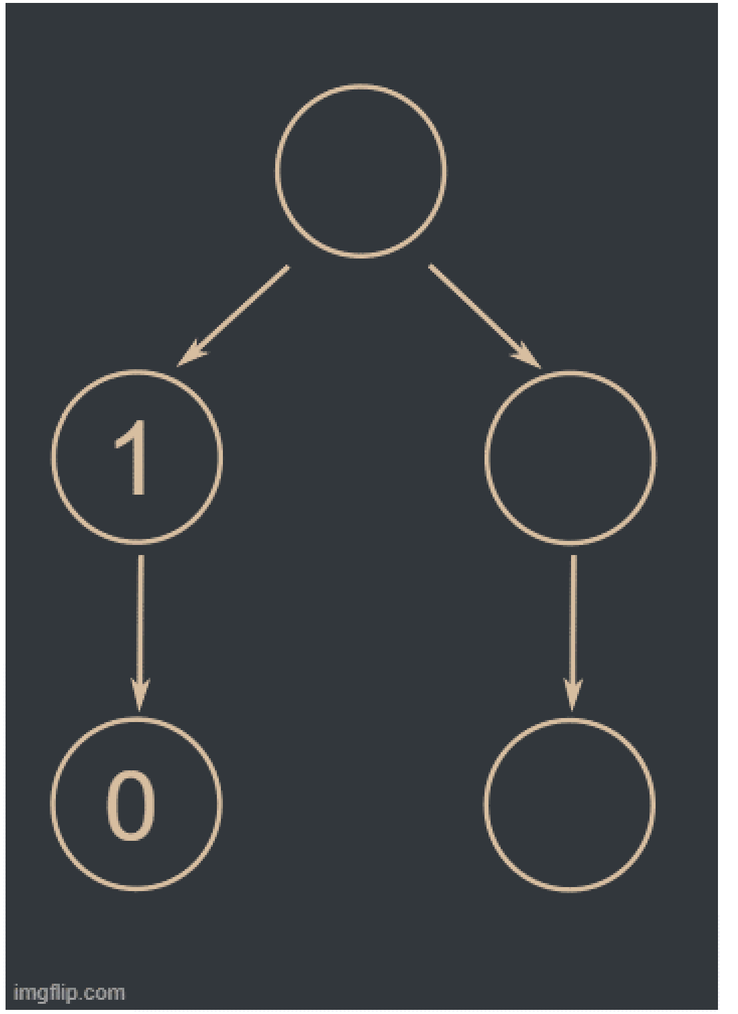




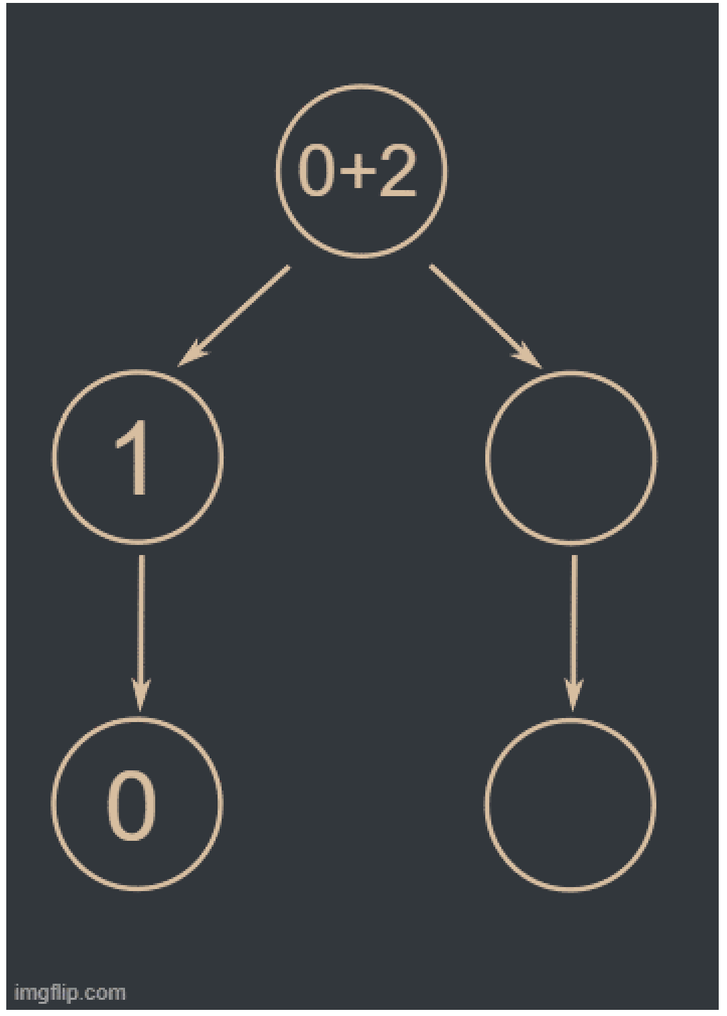
Real Case Tree : Subordinates

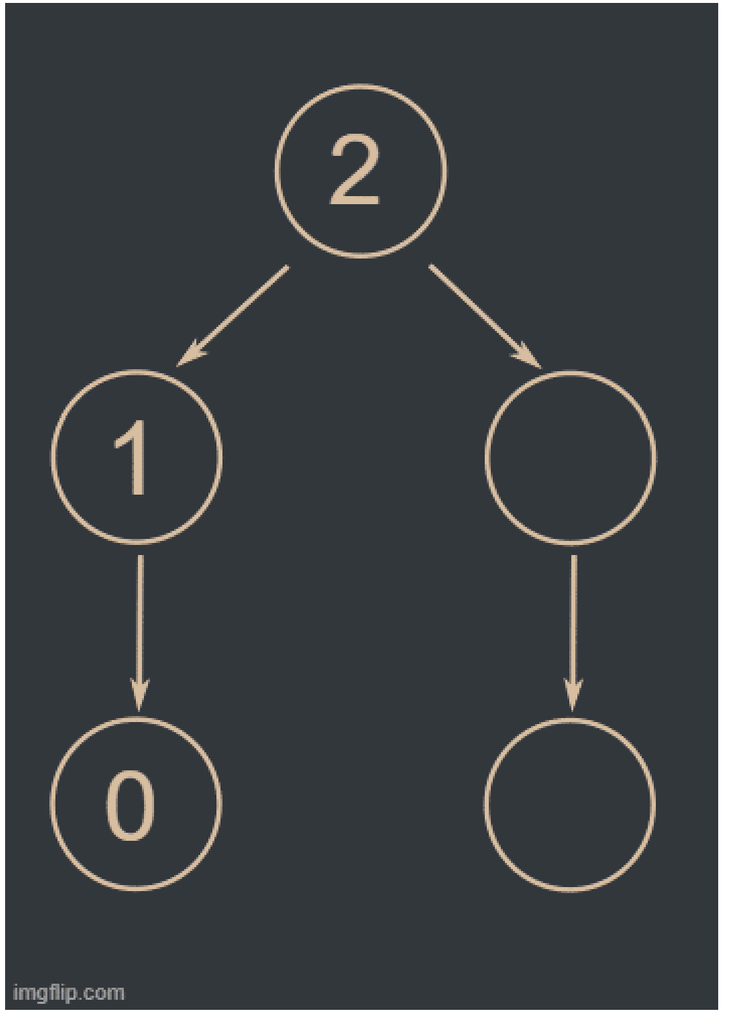


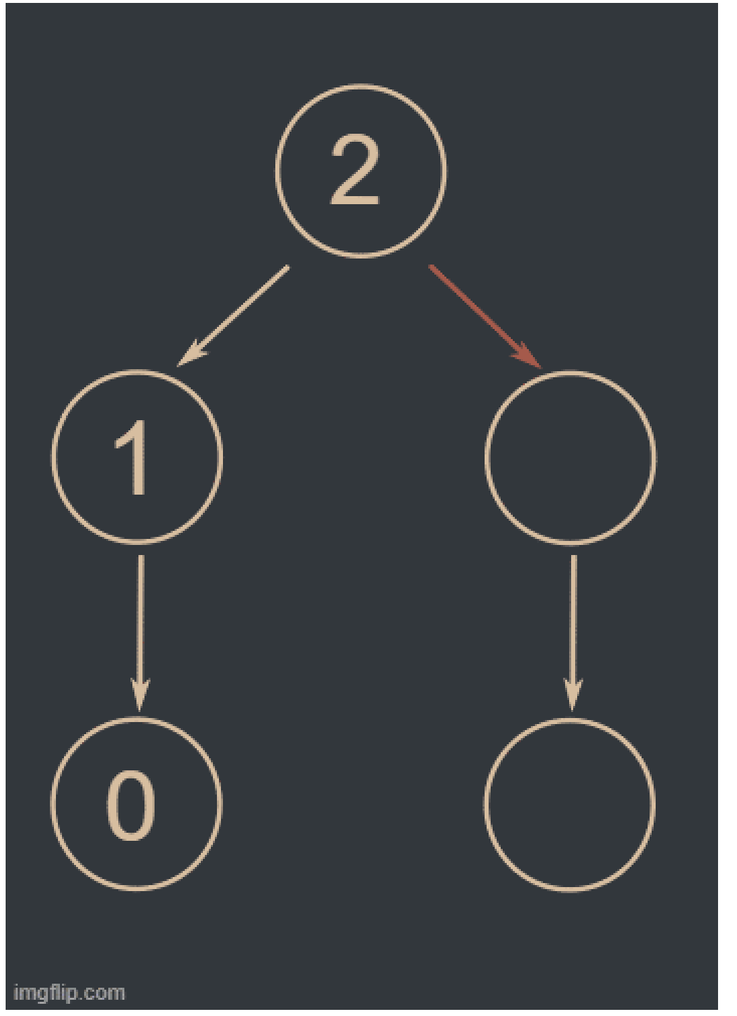




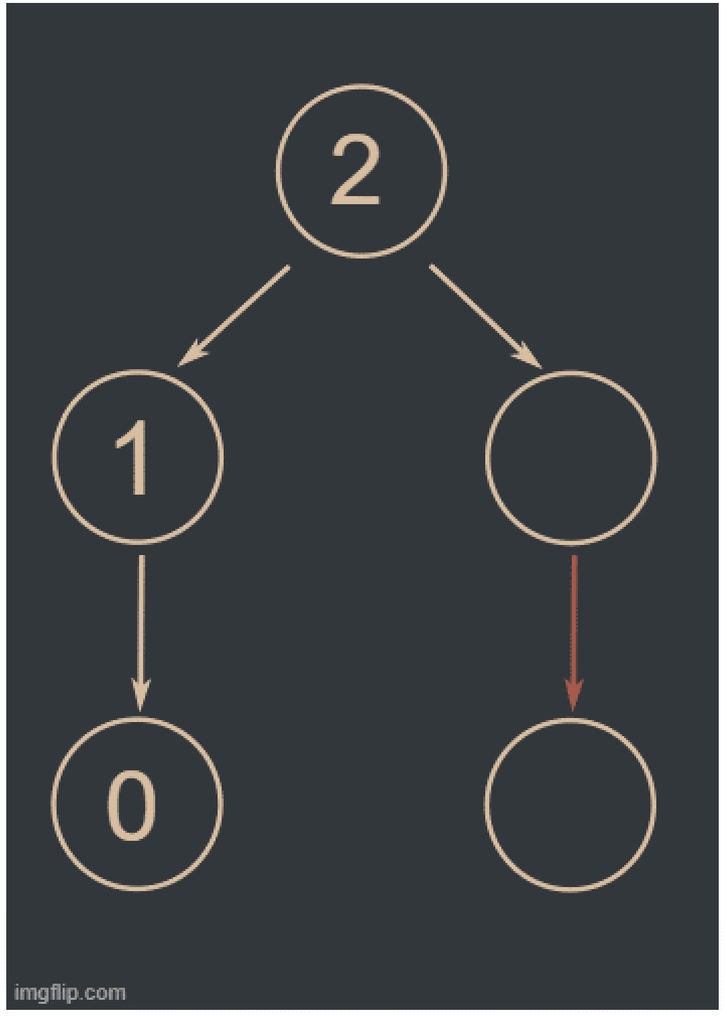
Real Case Tree : Subordinates

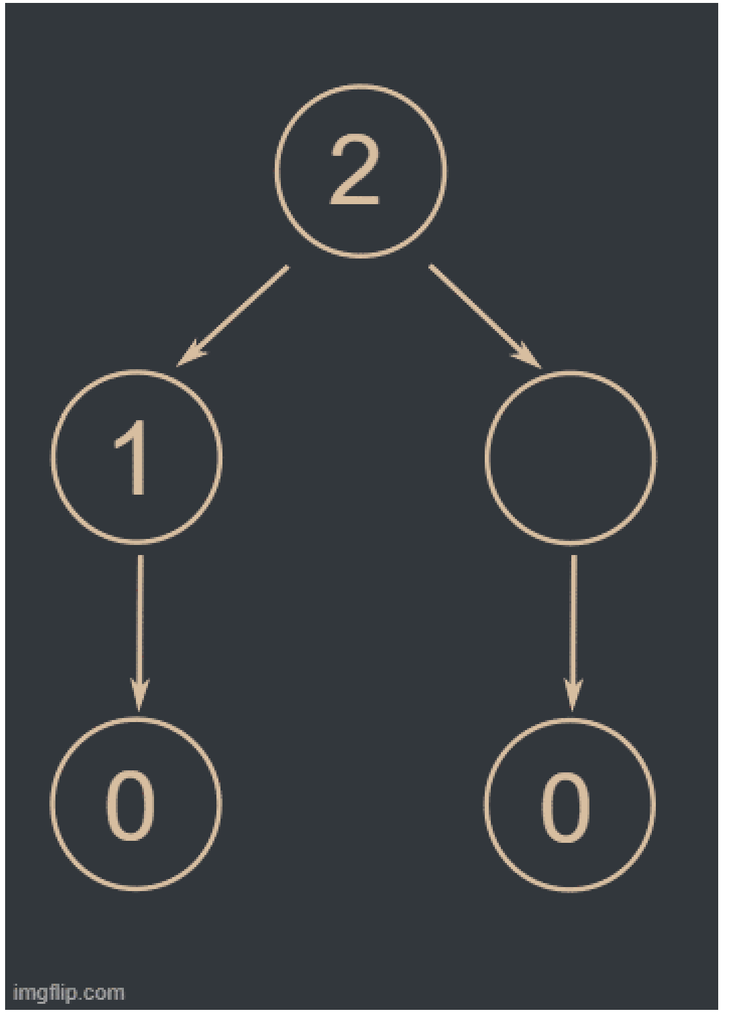


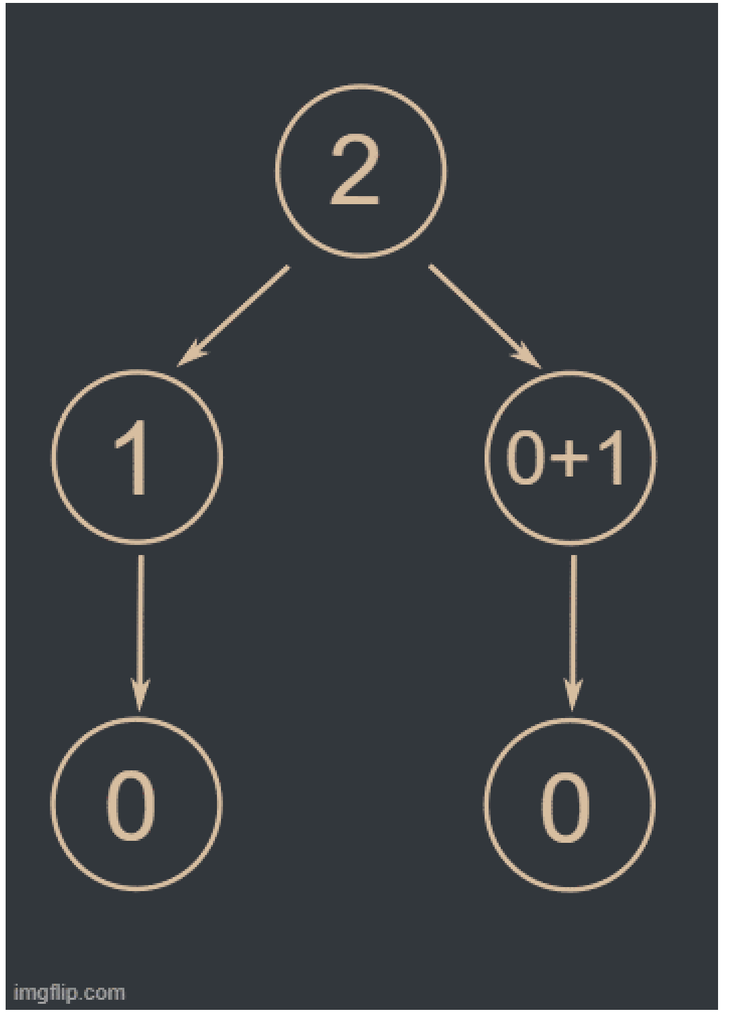




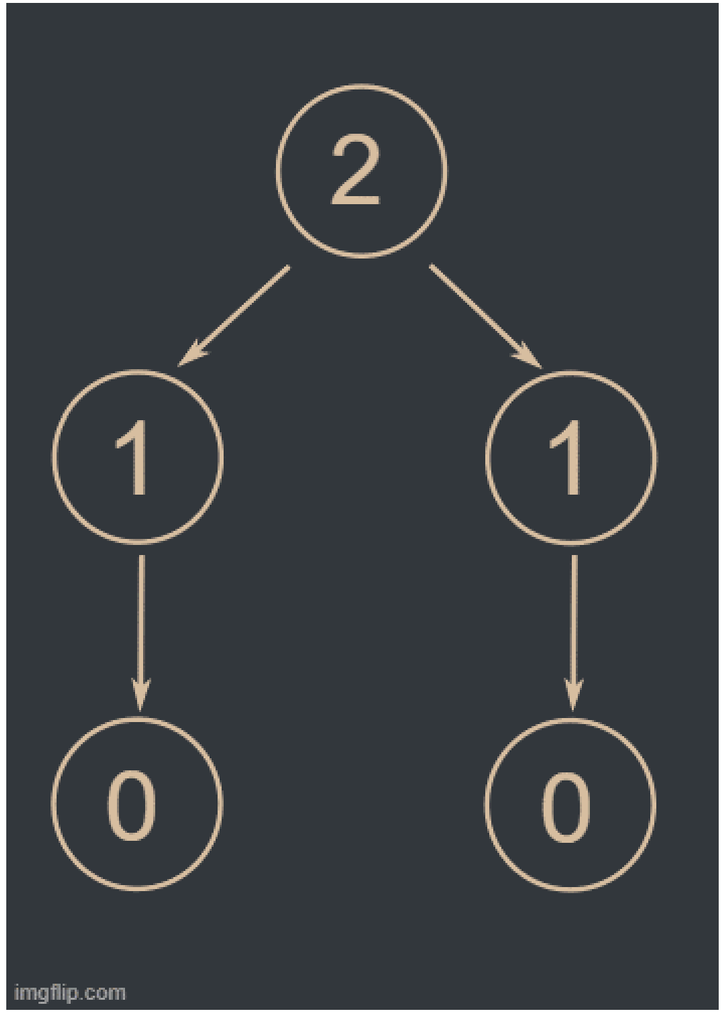
Real Case Tree : Subordinates

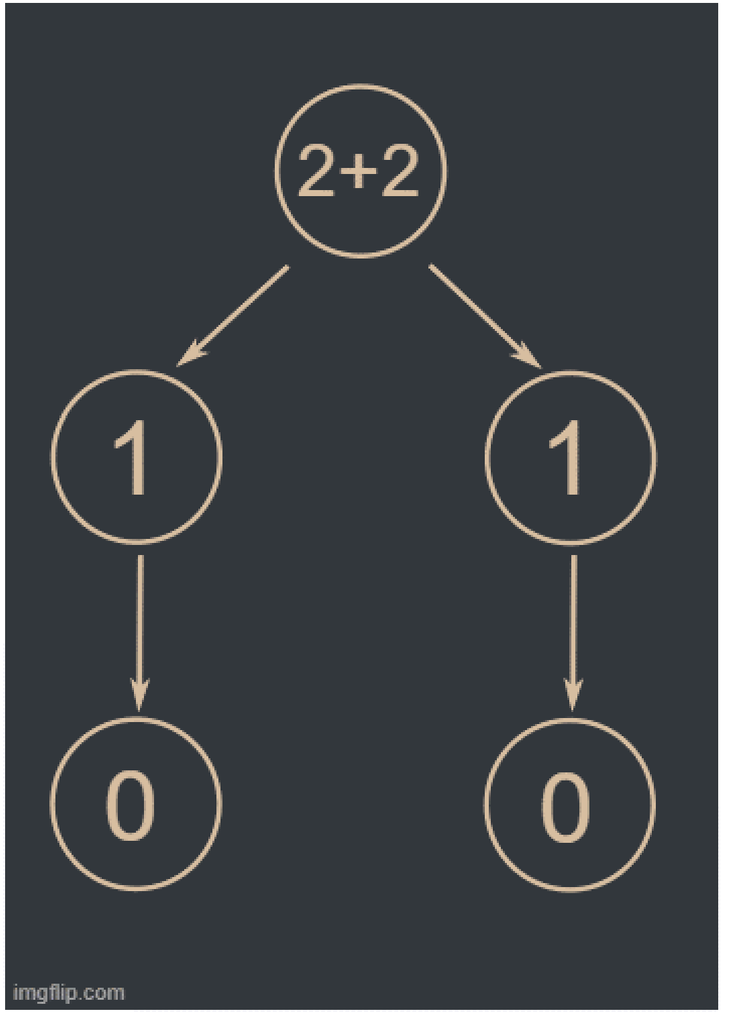


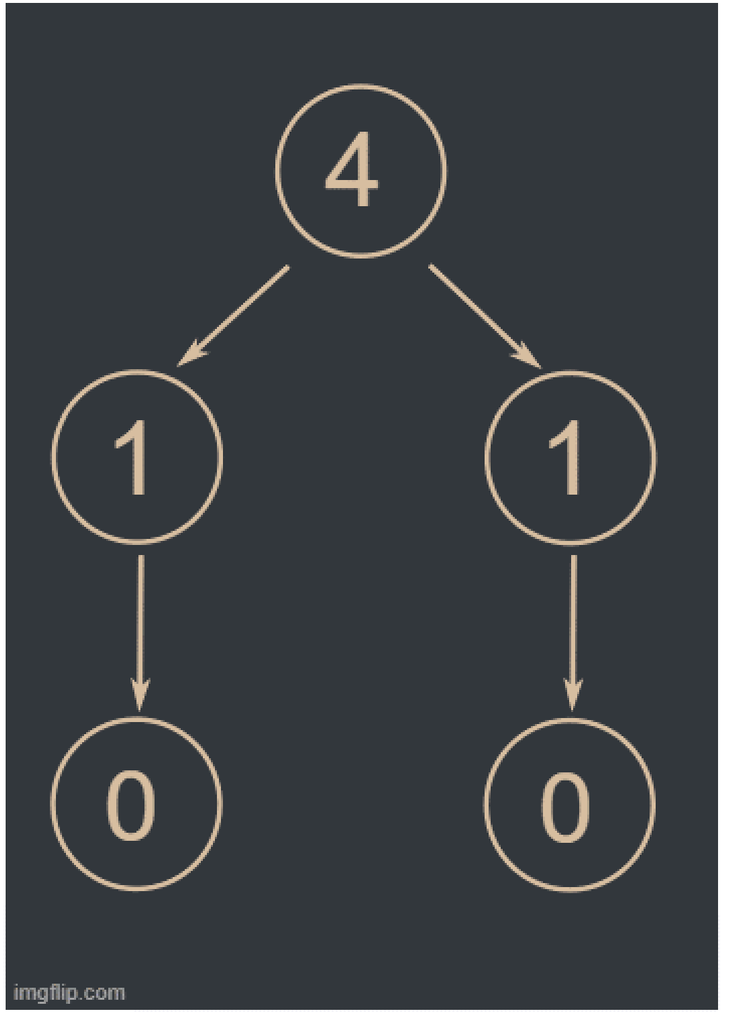




Real Case Tree : Subordinates







Real Case Tree : Subordinates

```cpp
#include <bits/stdc++.h>
using namespace std;

const int MAX = 2e5 + 7;

vector<int> children[MAX];
//menyimpan node-node karyawan

int jumlah_anak_buah[MAX];
//menyimpan jumlah anak buah yang dimiliki tipa node(karyawan)
```

Real Case Tree : Subordinates

```
12  int hitung(int karyawan){
13      jumlah_anak_buah[karyawan] = 0;
14      for(int anak_buah : children[karyawan]){
15          jumlah_anak_buah[karyawan] += hitung(anak_buah) + 1;
16      }
17      return jumlah_anak_buah[karyawan];
18  }
```

Real Case Tree : Subordinates

```
20  int main(){
21      int n;
22      cin >> n;
23      //input jumlah karyawan
```

Real Case Tree : Subordinates

```
20  int main(){
25      for(int i = 2; i <= n ; i++){
26          int bos;
27          cin >> bos;
28          children[bos].push_back(i);
29      }
30      // input bos karyawan 2 sd N
```

Real Case Tree : Subordinates

```
20  int main(){
32      hitung(1);
33
34      for(int i = 1; i <= n; i++){
35          cout << jumlah_anak_buah[i] << ' ';
36      }
37      // output jumlah anak buah tiap karyawan
38  }
```

Teknik Informatika | Kelas B

Terima Kasih

Ada pertanyaan?

Present by:
Ady Ulil Amri | Ramadani | Muhammad Fauzan Rusda | Taufiqurrahman Hendra | Muhammad Faaiq Fadhlurrahman