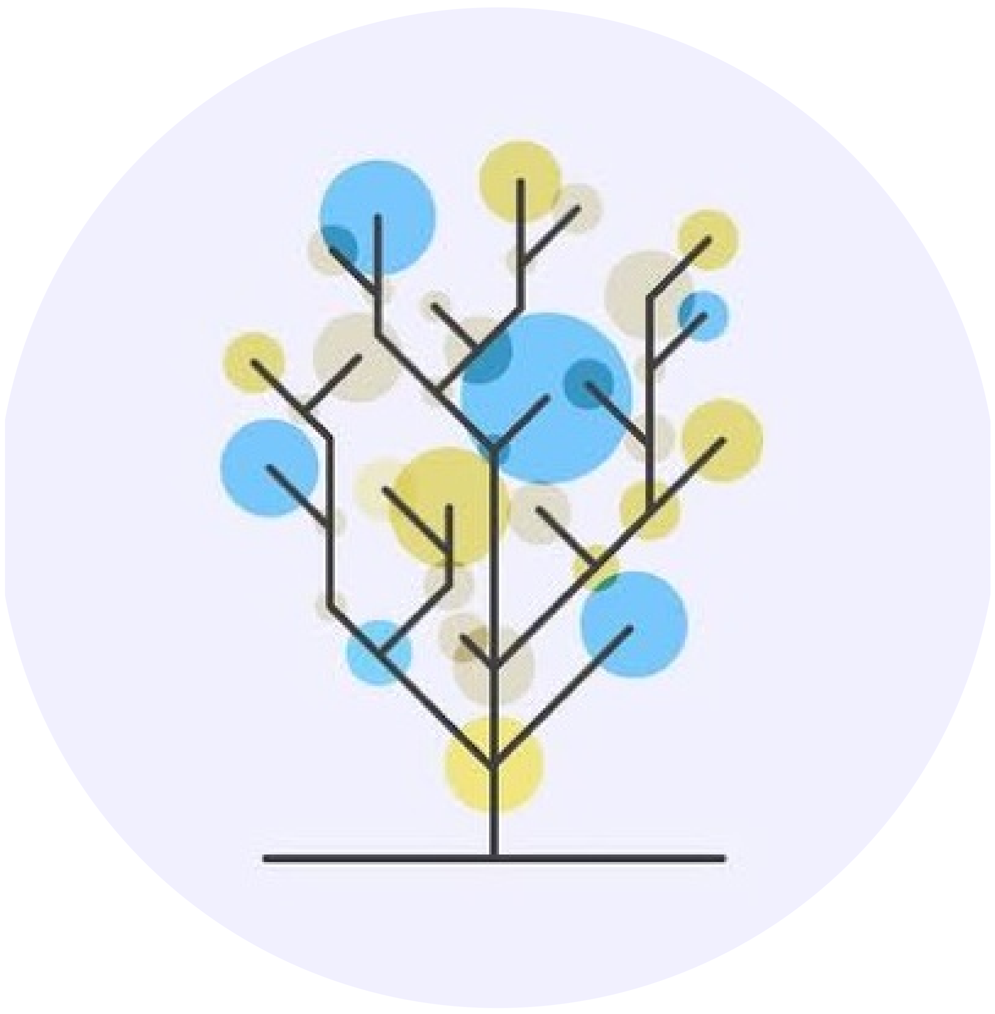

Graphs

04.01.2020

Data Structures & Algorithms

Intro

- Graphs are most used data structures when it comes to modeling real life.
- GRAPH IS SIMPLY A SET OF VALUES THAT ARE RELATED IN A PAIR WISE FASHION
- EACH NODE IS CALLED A **NODE** OR **VERTEX**.

-
  ```
  1 -- 3 --6
     /    /
    5 -- 4
          \
          11
  ```
- NODES ARE CONNECTED WITH **EDGES.**
- WE CAN USE GRAPHS TO REPRESENT MAYBE FRIENDSHIPS, MAYBE FAMILY TREES, NETWORKS, WORLD WIDE WEB, CITY CONNECTIONS.
- FACEBOOK USES IT FOR THEIR SOCIAL NETWORK, AMAZON USES IT FOR THEIR RECOMMENDATION ENGINES, GOOGLE MAPS USES IT TO DETERMINE THE SHORTEST PATH.
- TREES ARE TYPE OF GRAPH!

Types of Graphs

- There are many types of graphs.
- But there are certain characteristics to classify them:
  - `DIRECTED` `UNDIRECTED`
    ```
      * --> *                       * --- *
    ```
    - Facebook friendships are not one way then its unidirected.
    - Twitter is more directed because. I can follow people or people can follow me.
  - `WEIGHTED - UNWEIGHTED`
    - Nodes zaten her türlü data tutabiliyordu fakat weighted graphs ile edges de data tutabilir.

    -
      ```
            2       -11
        *  --->  * <---> *
      ```
    - Shortest path hesaplamada weighted graphs useful olabilir.
  - `CYCLIC - ACYCLIC`
    - When you have vertices connected in a circular fashion it is called a cyclic. By following nodes you can come back to the initial node.

    -
      ```
          *         *
         / \         \
        *___*      ___*
      ```

Representing Graphs

```
//*** GRAPHS ARE BUILT ON TOP OF OTHER DATA STRUCTURES.

//*** THERE ARE 3 DIFFERENT WAYS TO REPRESENT GRAPHS

//       2___0
//      / \
//     1___3

// YUKARIDAKİ GRAPH'İ 3 FARKLI ŞEKİLDE REPRESENT EDELİM:

//----------------------------------------------------------

//EDGE LIST
const graph = [ [0, 2],  [2, 3],  [2, 1], [1, 3] ];

//ADJACENT LIST
const graph  = [ [2] [2, 3] [0, 1, 3] [1, 2] ] //0. index 2'ye bağlı, 1. index 2 ve 3'e, 2. index
0,1,3, sonuncu index 1 ve 2'ye.
// HASH TABLE DA KULLANILABİLİR.

//ADJACENT MATRIX
const graph = {
  0: [0, 0, 1, 0],
  1: [0, 0, 1, 1],
  2: [1, 1, 0, 1],
  3: [0, 1, 1, 0]
} // 0 node'u yalnızca index 2 ile bağlı. 2 nodu'u tüm node'lar ile bağlı. Kendi node'unu hep 0
kabul ediyoruz.
```

Implementing a Graph

```
class Graph {
  constructor() {
    this.numberOfNodes = 0;
    this.adjacentList = {};
  }
  addVertex(node)  {
    this.adjacentList[node] = [];
    this.numberOfNodes++;
  }
  addEdge(node1, node2) {
    //uniderected Graph
    this.adjacentList[node1].push(node2);
    this.adjacentList[node2].push(node1);
  }
  showConnections() {
    const allNodes = Object.keys(this.adjacentList);
    for (let node of allNodes) {
      let nodeConnections = this.adjacentList[node];
      let connections = "";
      let vertex;
      for (vertex of nodeConnections) {
        connections += vertex + " ";
      }
      console.log(node + "-->" + connections);
    }
}
}

var myGraph = new Graph();
myGraph.addVertex('0');
myGraph.addVertex('1');
myGraph.addVertex('2');
myGraph.addVertex('3');
myGraph.addVertex('4');
myGraph.addVertex('5');
myGraph.addVertex('6');
myGraph.addEdge('3', '1');
myGraph.addEdge('3', '4');
myGraph.addEdge('4', '2');
myGraph.addEdge('4', '5');
myGraph.addEdge('1', '2');
myGraph.addEdge('1', '0');
myGraph.addEdge('0', '2');
myGraph.addEdge('6', '5');

myGraph.showConnections();
```

Pros and Cons

```
    PROS                          CONS
  --------                      -------
```

`Relationships` `Scaling is hard`

- Very useful when it comes to relationships. Some data is need to be in graph form there is no other way.
- Scaling is very hard.
- There are tools to build complex graphs.

More blogs