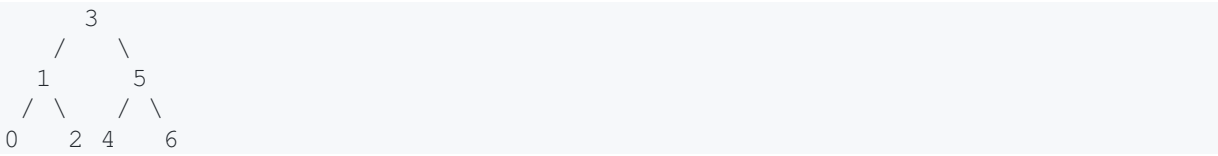
Binary trees

A binary tree is a tree whose every node either branches to two binary trees or is a leaf, i.e. contains a value. Here is an example of a binary tree



- (**A**) Design a JSON Schema for representing binary trees. Both the branching nodes and leafs should carry integer values. An important property is that each node in the tree either has both a left and a right subtree, or no subtree at all (not just a left subtree for instance).
    - Variant: Make one representation using arrays and one using key/values for children.
- (**B**) Show a JSON element representing the above example tree, and which is valid according to your Schema.
    - Variant: Make values in branches optional (but values in leafs should still be required)
- (**C**) Write a JSONPath query that returns all leaf elements of a binary tree. For the above example, it should return 0,1,2,3,4,5,6 (in any order).
    - Variant: Find all data values of left subtrees (0,1,4)
    - Variant: Find all values in the third level of the tree (0,2,4,6)
    - Variant: Find all values greater than 3
    - Variant: Find all values greater than the value in the root node (should work for all trees)

Flights

Given the following schema:

```
Airports(_code,city)

FlightCodes(_code, airlineName)

Flights(departureAirport, destinationAirport, _code)
  departureAirport -> Airports.code
  destinationAirport -> Airports.code
  code -> FlightCodes.code
```

Write a JSON Schema corresponding to this database schema. Translate the relational schema as faithfully as possible (there is nothing you can do about the references, but can you have primary keys in JSON?). Also write a JSON document with the data in the table below, which validates with your schema.

Hint: You can use the "additionalProperties" keyword to specify a schema for all properties of an object (except the ones mentioned in "properties").

Flight code	Airline	Dep.city	Dep.airport	Arr.city	Arr.airport
SK111	SAS	Gothenburg	GOT	Frankfurt	FRA
AF222	Air France	Paris	ORY	Malta	MLA
AB222	Air Berlin	Frankfurt	FRA	Munich	MUC
KM111	Air Malta	Munich	MUC	Malta	MLA

Applications

Below is some JSON data. It has been compiled by translating this schema in the most direct way possible:

Applicants(appNum, name)

Choices(applicant, code, choiceNum, meritScore)
applicant -> Applicants.appNum


```
{
  "Applicants": [
    {"appNum":"a1", "name": "Andersson"},
    {"appNum":"a2", "name": "Jonsson"},
    {"appNum":"a3", "name": "Larsson"}
  ],

  "Choices": [
    {"applicant":"a1","code":"MPSOF","choiceNum":1,"meritScore":750},
    {"applicant":"a1","code":"MPALG","choiceNum":2,"meritScore":750},
    {"applicant":"a1","code":"MPCSN","choiceNum":3,"meritScore":800},
    {"applicant":"a2","code":"MPALG","choiceNum":1,"meritScore":700},
    {"applicant":"a3","code":"MPCSN","choiceNum":1,"meritScore":850},
    {"applicant":"a3","code":"MPALG","choiceNum":2,"meritScore":850}
  ]
}
```


(A) Can you rewrite the data into a more semi-structured format that uses the fact that there are no tables? Here are some suggestions:

- Avoid repeating applicant numbers (and the implicit references that exist in the data)?
- Use key/value pairs instead of an array of rows?
- Maybe choice numbers are not needed?

(B) Write a JSON Schema for your modified data.

(C) Write a JSONPath query on your modified data that finds all "choices" where choiceNum is 1 and meritScore is greater than 800.