Laboratorio I
a.a. 2025/2026

Alberi

Contenuti

- Codifica di alberi con oggetti
- Alberi binari
  - Esercizi
- Alberi *k*-ari
  - Esercizi

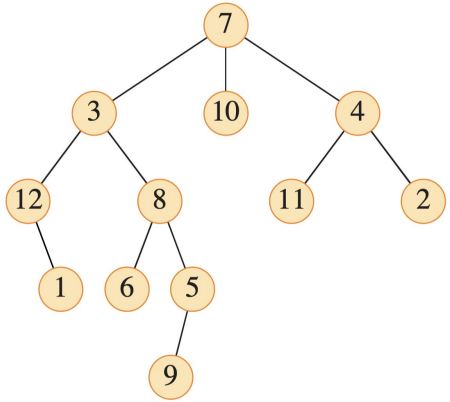
Codifica degli alberi

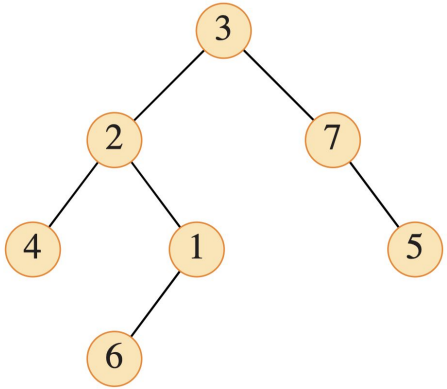
Avete visto (a Fondamenti) che un **albero** è un **grafo connesso aciclico**. I nodi di grado 1 sono detti **foglie**, i nodi di grado > 1 sono detti **nodi interni.**

Un **albero radicato** ha un nodo identificato come **radice.**

Ogni nodo interno ha 1 o più **figli**, che consideriamo ordinati (e numerati 0...*k*).

Nel caso particolare di *k*=2, si parla di **albero binario**; in questo caso, chiameremo i figli **sinistro** (sx) e **destro** (dx).





Codifica degli alberi

Concentriamoci sugli **alberi binari**, e associamo ad ogni nodo un qualche **valore**.

Come possiamo rappresentare un nodo in JavaScript?

Codifica degli alberi

Concentriamoci sugli alberi binari, e associamo ad ogni nodo un qualche **valore**.

Possiamo allora rappresentare un nodo con un oggetto JavaScript con tre chiavi

- `val`: il valore associato al nodo
- `sx`: il figlio sinistro del nodo (chiave assente o `null` se il figlio sx manca)
- `dx`: il figlio destro del nodo (chiave assente o `null` se il figlio dx manca)

```
{ val: 5, sx: …, dx: … }
```

Ma come possiamo indicare i figli?

Codifica degli alberi

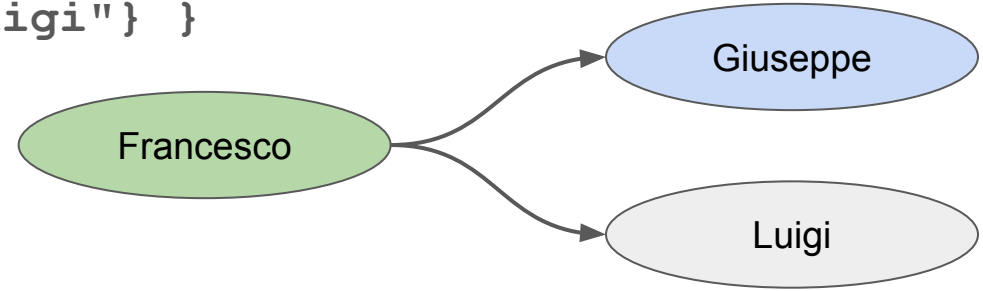
I figli di un nodo sono anch'essi nodi… quindi usiamo la stessa codifica!

```
let pino={val: "Giuseppe"}, gigi={val: "Luigi"}

let ciccio={val: "Francesco", sx: pino, dx: gigi}
```

Oppure, direttamente:

```
let ciccio={val: "Francesco", sx: {val: "Giuseppe"},
            dx: {val: "Luigi"} }
```

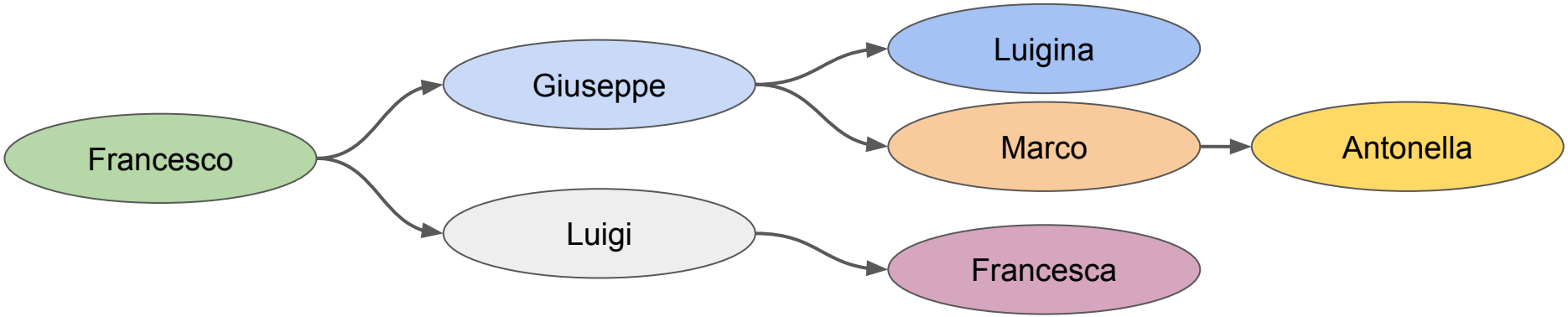


Codifica degli alberi

Gli alberi sono naturalmente una struttura **ricorsiva**.

Abbiamo nodi, i cui figli sono nodi, i cui figli sono nodi … fino ad arrivare a nodi senza figli.

Se sappiamo come manipolare un nodo e i suoi figli per fare qualcosa… allora sappiamo come manipolare un albero intero, di qualunque dimensione!



Alberi binari a valori numerici

- Sappiamo che in JavaScript le chiavi devono essere stringhe o numeri, ma i loro valori possono essere di qualunque tipo JavaScript
- Il valore (val) associato a un nodo potrebbe dunque essere qualunque cosa:
  - Una stringa, come “Francesco” o “Luigina”
  - Un numero, come 17 o 3.1415926
  - Un booleano
  - Una funzione
  - Un array
  - Un oggetto
- Giusto per fare un po’ di pratica, considereremo gli alberi binari con valori numerici

Esercizi

- Trovare il massimo fra i valori in un albero binario
- Trovare la somma dei valori in un albero binario
- Dire se un albero binario contiene un valore cercato o no
- Contare quanti sono i nodi di un albero binario che hanno un valore dato
- Scambiare fra di loro i rami destro e sinistro della radice di un albero binario
- Tagliare da un albero binario tutti i rami che iniziano da un nodo con valore dato

Alberi *k*-ari a valori numerici

Ora che sappiamo manipolare gli alberi binari (o 2-ari), possiamo spingerci oltre e considerare gli alberi *k*-ari, con *k*>2.

Non possiamo più dare un *nome* ai figli (dx e sx)... ora potremmo avere un numero qualunque di figli.

Idee?

Alberi *k*-ari a valori numerici

Ora che sappiamo manipolare gli alberi binari (o 2-ari), possiamo spingerci oltre e considerare gli alberi *k*-ari, con *k*>2.

Non possiamo più dare un *nome* ai figli (dx e sx)... ora potremmo avere un numero qualunque di figli.

Fortunatamente, JavaScript ha giusto un tipo di dato che fa al caso nostro:

```
{ val: 5, figli: [ … ] }
```

Esercizi

Esercizi simili ai precedenti, ma questa volta su alberi *k*-ari

- Trovare il massimo fra i valori in un albero
- Dire se un albero contiene un valore cercato o no
- Applicare una funzione data a tutti i valori contenuti in un albero, sostituendo in ogni nodo il valore attuale con il risultato della funzione applicata al valore attuale

Alberi *k*-ari a valori arbitrari

Possiamo sfruttare il fatto che il valore associato a un nodo può essere di qualunque tipo (anche diverso da nodo a nodo) per creare strutture interessanti

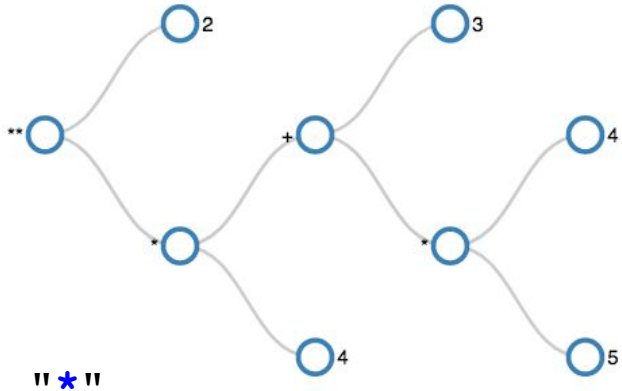
```
{ val: " … ", figli: [ … ] }

{ val: { … }, figli: [ … ] }

{ val: [ … ], figli: [ … ] }
```

Esempio: valutazione di espressioni

Immaginiamo un albero in cui i nodi intermedi sono operatori (aritmetici) e le foglie sono operandi (numerici)




```
{
  val: "**",
  sx: { val: "*",
        sx: { val: 4},
        dx: { val: "+",
              sx: {  val: "*",
                     sx: { val: 5 },
                     dx: { val: 4 }
                  },
              dx: { val: 3 }
            }
  },
  dx: { val: 2 }
}
```

(4*((5*4)+3))2**

Si scriva una funzione che, dato un albero così fatto, restituisca il valore dell'espressione rappresentata dall'albero

Q & A