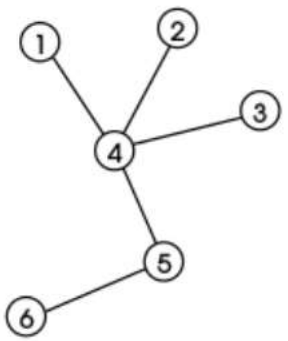
C8

miercuri, 24 ianuarie 2024 23:24

Arbori

Un arbore e un *graf neorientat conex și fără cicluri*.

Arborii reprezintă grafurile cele mai simple ca structură din clasa grafurilor conexe, ei fiind și cei mai frecvent utilizați în practică.



http://en.wikipedia.org/wiki/File:Tree_graph.svg

3

- conex = drum între orice 2 noduri (din 1 sau mai mulți pași)

E compus din *noduri* și *ramuri* (muchii).

Un arbore cu *n* noduri are *n* - 1 ramuri

Exemple de arbori

Exemple de arbori

(i) - arbore

(ii) - arbore

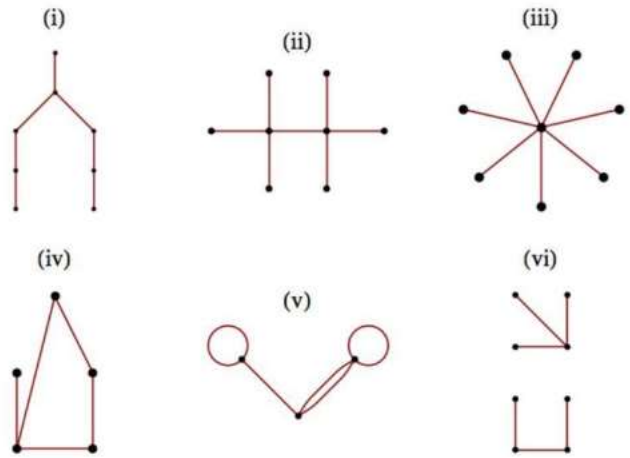
(iii) - arbore

(iv) - nu e arbore

(v) - nu e arbore

(vi) - nu e arbore

(e pădure)



Applied Discrete Structures, Al Doerr, Ken Levasseur, University of Massachusetts Lowell, 2022 5

Un tip de graf strâns legat de conceptul de arbore, dar care nu îndeplinește toate condițiile unui arbore e *pădurea*.

O *pădure* este un graf neorientat neconex a cărui componente conexe sunt *arbori*.

Dacă avem graful G = (V, E) neorientat și fără cicluri, iar |V| = n, propozițiile următoare sunt echivalente:

Dacă avem graful G = (V, E) neorientat și fără cicluri, iar |V| = n, propozițiile următoare sunt echivalente:

- G e un *arbore*
- Pentru fiecare 2 noduri distincte din V, există *un singur drum* între ele
- G este conex, și dacă avem o muchie e, atunci graful (V, E – {e}) *nu e conex*
- G nu conține cicluri, dar *dacă adăugăm o muchie* în plus vom avea un ciclu
- G este conex, iar *|E| = n-1*

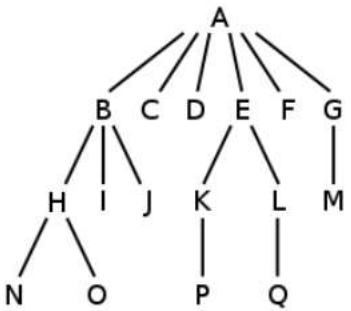
Arbore cu rădăcină

De obicei identificăm un nod anume numit *rădăcina*, și *orientăm* muchiile în același sens față de rădăcină

Orice nod în afară de rădăcină are un unic *părinte*

Un nod poate avea mai mulți *copii* (*fii*)

Nodurile fără copii se numesc noduri *frunză*



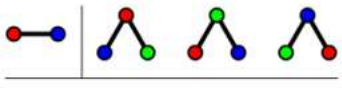
Imagine: http://en.wikipedia.org/wiki/File:N-ary_to_binary.svg

8

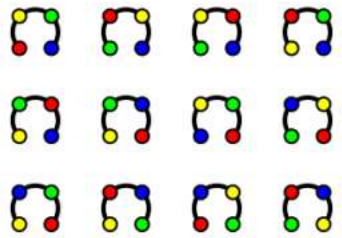
- Arborii sunt un mod natural de a reprezenta *structuri ierarhice:*
  - *sistemul de fișiere* (subarborii sunt cataloagele)
  - *arborele sintactic* într-o gramatică (ex. expresie)
  - *ierarhia de clase* în programarea orientată pe obiecte
  - *fișierele XML* (elementele conțin alte elemente)

Arbori ordonați și neordonați

Ordinea dintre copii poate conta (ex. arbore sintactic) sau nu

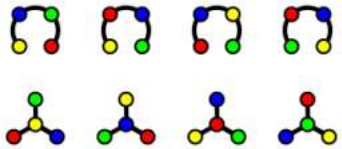


Arborii neordonați cu 2 – 4 noduri – în figură:



Există n^{n-2} *arbori neordonați cu n noduri*

Există n^{n-2} *arbori neordonați* cu *n* noduri (*formula lui Cayley*)



Imagine: http://en.wikipedia.org/wiki/File:Cayley's_formula_2-4.svg

10

Arbori binari

Într-un arbore binar, fiecare nod are cel mult doi copii, identificați ca fiul stâng și fiul drept (oricare/ambii pot lipsi)

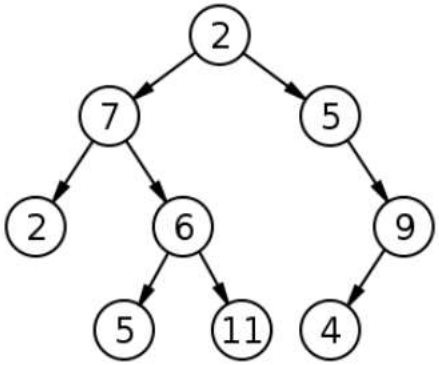
⇒ un arbore binar e:

- arborele vid sau
- un nod cu cel mult doi subarbori

Arbori binari

Un arbore binar de înălțime *n* are cel mult $2^{n+1} - 1$ noduri

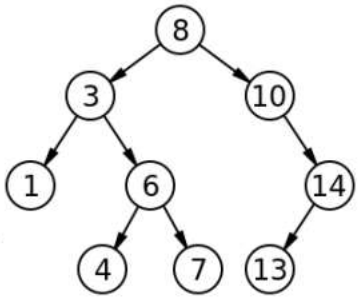
$2^{h+1} - 1$ noduri



13

Arbori binari de căutare

Arborii binari de căutare sunt arbori binari care memorează valori sortate *în ordine*.



Pentru fiecare nod,
relativ la valoarea din rădăcină:
- subarborele *stâng* are valori *mai mici*
- subarborele *drept* are valori *mai mari*

Căutarea se face *recursiv*, comparând mereu elementul căutat cu *rădăcina* subarborelui curent:
- dacă sunt egale *am găsit* elementul în arbore
- dacă e < rădăcina curentă, se continuă căutarea în subarborele stâng
- dacă e > rădăcina curentă, se continuă căutarea în subarborele drept

14

- dacă e > rădăcina curentă, se continuă căutarea în subarborele drept

14

Arborii de căutare pot fi folosiți la *sortarea unui șir de obiecte* care pot fi ordonate.

Mai întâi se *creează arborele* de căutare cu obiectele din șir:

- primul obiectva fi rădăcina arborelui
- următoarele obiecte se introduc în subarborele stâng sau drept, în funcție de valoare

Iar apoi *se parcurge arborele de căutare în inordine* (arbore stâng, rădăcină, arbore drept) și vom obține obiectele din șir ordonate.

15

Parcurgerea arborilor

în *preordine*: *rădăcina*, subarborele *stâng*, subarborele *drept*

în *inordine*: subarborele *stâng*, *rădăcina*, subarborele *drept*

în *postordine*: subarborele *stâng*, subarborele *drept*, *rădăcina*

subarborii se parcurg și ei tot în ordinea dată (pre/in/post ordine)!

Pentru expresii, obținem astfel formele prefix, infix și postfix

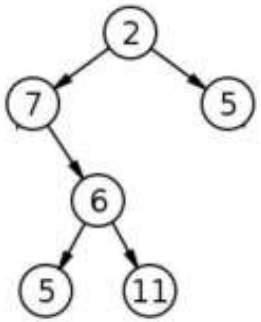
Parcurgerea în pre-/ post-ordine e definită la fel pentru orice arbori

Pentru expresii, obținem astfel formele prefix, infix și postfix

Parcurgerea în pre-/ post-ordine e definită la fel pentru orice arbori (nu doar binari).

Parcurgerea în preordine

```
def rsd(tree):
  if (tree != None):
    return [tree["value"]] + rsd(tree["left"]) + rsd(tree["right"])
  else:
    return []
```

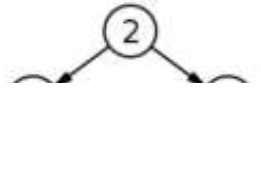



```
print(rsd(binary_tree))
# [2, 7, 6, 5, 11, 5]
```

26

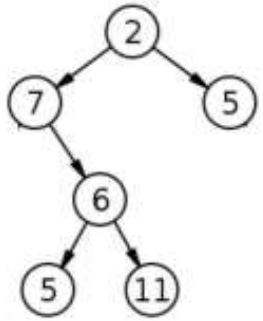
Parcurgerea în inordine

```
def srd(tree):
  if (tree != None):
    return  srd(tree["left"]) + [tree["value"]] + srd(tree["right"])
  else:
    return []
```



```
    return []


print(srd(binary_tree))
# [7, 5, 6, 11, 2, 5]
```

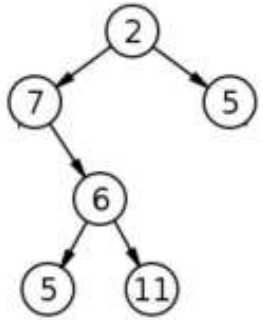


27

Parcurgerea în postordine

```
def sdr(tree):
    if (tree != None):
        return  sdr(tree["left"]) + sdr(tree["right"]) + [tree["value"]]
    else:
        return []


print(sdr(binary_tree))
#[5, 11, 6, 7, 5, 2]
```



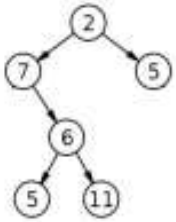
28

Adaugarea unui nod nou

Adaugarea unui nod nou

Adăugarea unui nod nou la un părinte și o poziție anume:

```
def adaugare_nod_pozitie(parinte, nod_nou, pozitie):
    if (parinte[pozitie] == None):
        parinte[pozitie] = nod_nou
    return parinte
```

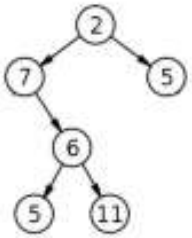



```
binary_tree["left"]=adaugare_nod_pozitie(binary_tree["left"],
{"value": 100, "left": None, "right": None}, "left")
print(rsd(binary_tree))
#[2, 7, 100, 6, 5, 11, 5]
```

Adaugarea unui nod nou

Adăugarea unui nod nou *în arbore binar de căutare*:

```
def adaugare_nod(tree, nod_nou):
    if (tree == None):
        return nod_nou
    if (nod_nou["value"]<tree["value"]):
        tree["left"] = adaugare_nod(tree["left"], nod_nou)
    else:
        tree["right"] = adaugare_nod(tree["right"], nod_nou)
    return tree
```




```
print(rsd(adaugare_nod(binary_tree,{"value": 1, "left": None,
"right": None})))
#[2, 7, 1, 6, 5, 11, 5]
```

```
print(rsd(adaugare_nod(binary_tree,{"value": 1, "left": None,
"right": None})))
#[2, 7, 1, 6, 5, 11, 5]
```

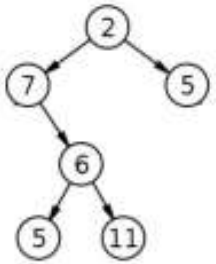
30

Ștergerea unui nod/ subarbore

Ștergerea unui nod (sau subarbore) de la un anumit părinte dat ca parametru:

```
def stergere_nod(parinte, valoare_nod):
    if (parinte["left"]["value"] == valoare_nod):
        parinte["left"] = None
    elif(parinte["right"]["value"] == valoare_nod):
        parinte["right"] = None

stergere_nod(binary_tree, 5)
print(rsd(binary_tree))
#[2, 7, 6, 5, 11]
```



Tupluri în PYTHON

Un *tuplu* este o colecție de date predefinite în PYTHON (pe lângă liste, mulțimi și dicționare).

Un tuplu este o colecție de date *ordonată* și *nu*

Un tuplu este o colecție de date *ordonată* și *nu se mai poate schimba după creare*.

Un tuplu se scrie între paranteze rotunde:

tuplu = (2, 5, 7, 1, 5)

32

Tupluri în PYTHON

Elementele unui tuplu:

- sunt *ordonate* (se pot accesa prin index pozitiv sau negativ)
- *nu se mai pot schimba* după creare
- permit *duplicate*

33

Tupluri în PYTHON

Numărul de elemente din tuplu se poate afla cu funcția *len()*

```
a = (1, 6, 8)
print(len(a)) # 3
```

Pentru a crea un tuplu cu un singur element e nevoie să se pună *între paranteze rotunde și o virgulă la final*:

```
tuplu = (5,)
```

34

Tupluri în PYTHON

Se poate crea un tuplu și cu constructorul *tuple()*

```
a = tuple((4, 6, 8))
b = tuple(["Arad", "Timisoara"])
```

Elementele se accesează prin indecși:

```
print(a[0])          # 4
print(b[1])          # Timisoara
```

```
print(a[0])          # 4
print(b[1])          # Timisoara
print(b[-2])         # Arad
print(a[1:3])        # (6, 8)
print(9 in a)        # False
print(8 in a)        # True
```

35

Tupluri în PYTHON

Nu se pot adăuga elemente în tuplu și nici *nu se pot șterge* ulterior creerii acestuia.

Aceste operații sunt permise la lucrul cu liste. Dacă vrem să creem un nou tuplu cu elemente diferite se poate transforma în listă și prelucra:

```
a = (3, 5, 7, 3)
lista = list(a)
#prelucrarea listei lista
```

36

Tupluri în PYTHON

Putem *extrage elemente* din tuplu și apoi să le prelucrăm independent:

Putem *extrage elemente* din tuplu și apoi să le prelucrăm independent:

```
tuplu = (3, 3, 6, 8)
a, b, c, d = tuplu
print(a)                # 3
print(b)                #3
print(c)                #6
```

Dacă numărul elementelor din tuplu e mai mare e obligatoriu să folosim un asterisc la ultimul obiect:

```
a, b, *c = tuplu        # c = (6, 8)
```

37

Tupluri în PYTHON

Se poate crea un tuplu nou cu elementele din alte 2 sau mai multe tupluri:

```
a = (1, 2, 3)
b = ("a", "b", "c")
c = a + b

print(c)                # (1, 2, 3, "a", "b", "c")
```