ch 5: ağaçlar

todo

5.1 hedefler

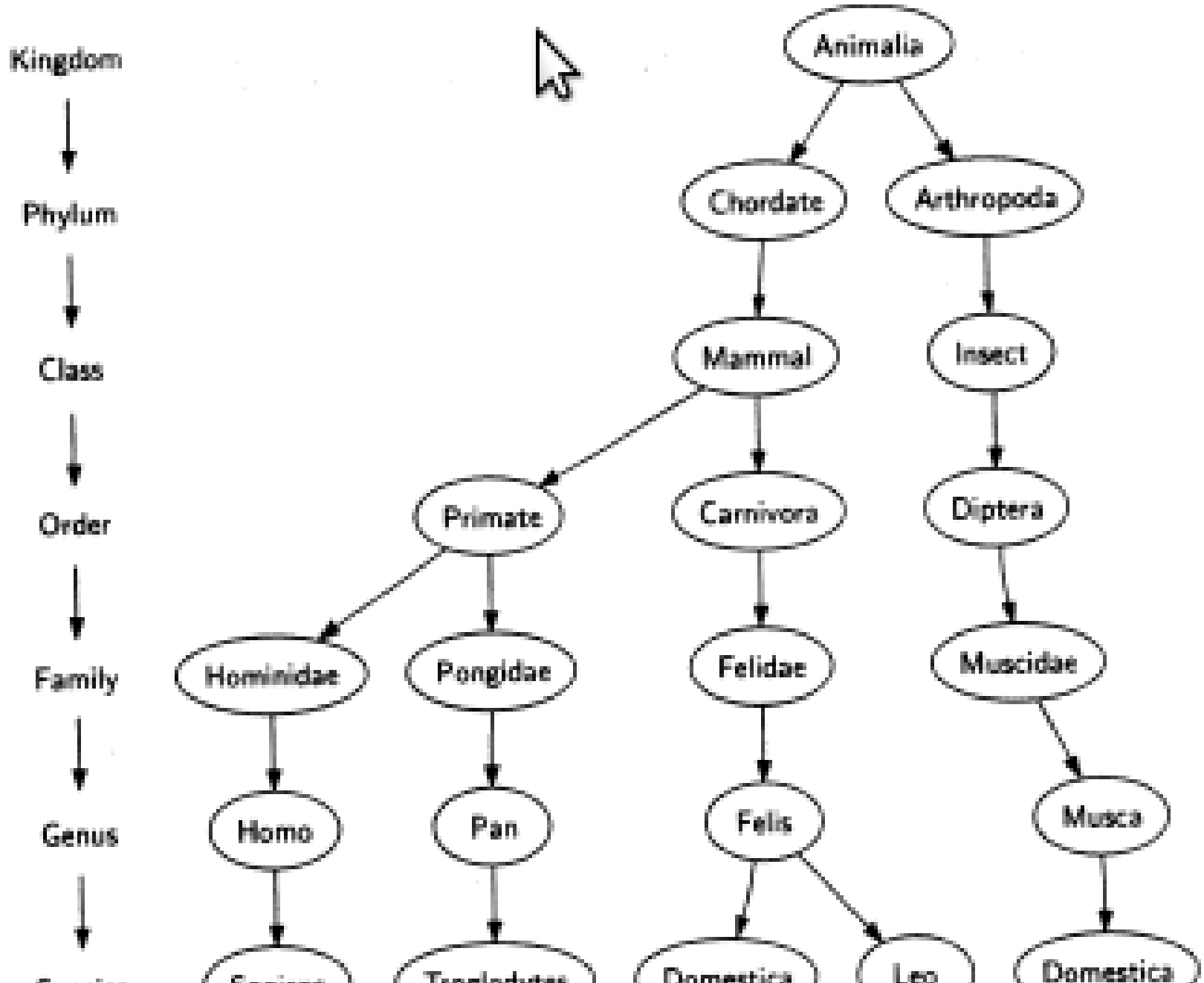
- → ağaç veri yapısı: tanımla, kullan
- → harita veri yapısı için ağaç kullanımı
- → listeyle ağaç gerçekleme
- → sınıf ve referansları kullanarak ağaç gerçekleme
- → özyineli veri yapısı olarak ağaç gereçekleme
- → yığın (heap) kullanarak öncelikli kuyruk

5.2 ağaç örnekleri

- → işletim sistemleri, grafikler, veritabanı sistemleri, bilgisayar ağları
- → botanik kuzeniyle ortak özelliklere sahip: ağaç, kök, dal, yaprak
- → ağaç veri yapısında kök yukarıdadır, yapraklar aşağıdadır

örnek

→ biyolojik sınıflandırma ağacı



örnek

dosya sistemleri ağacı

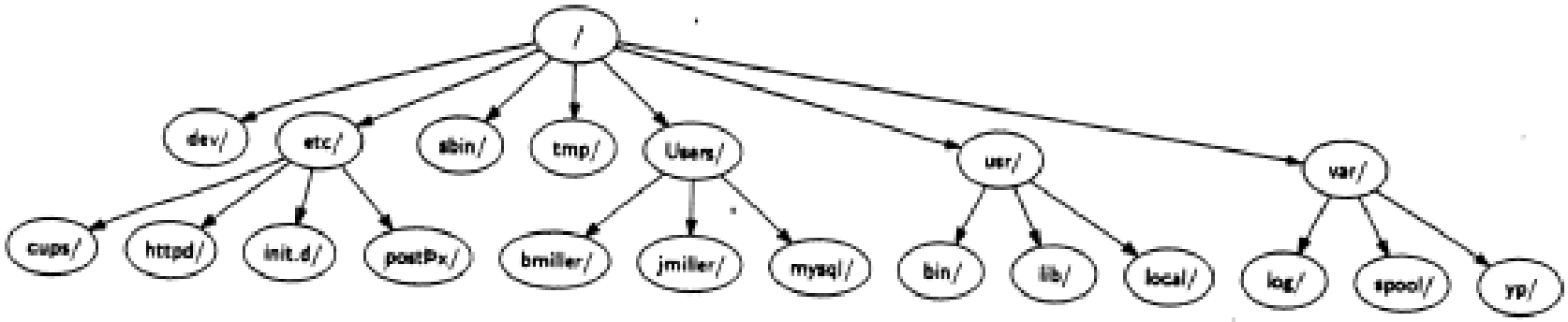


Figure 5.2: A Small Part of the Unix File System Hierarchy

→ klasörler ağaçlar yapısıyla tutulur

→ bir klasörü (ve altındakileri) başka bir konuma taşımak

→ altağacı (**subtree**) yeni dala (**branch**) taşımaktır

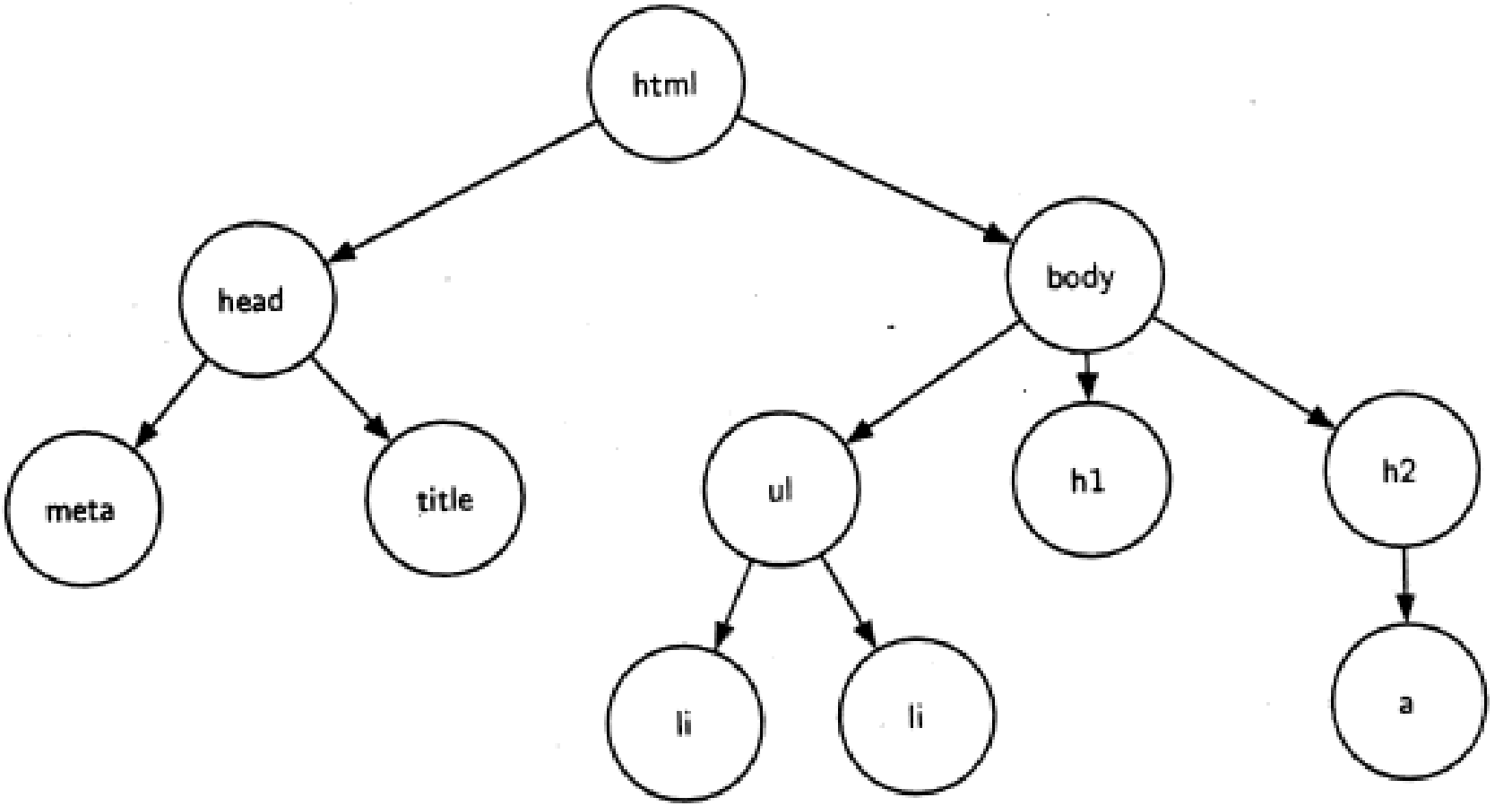
örnek

web sayfası ağacı

```
1   <html xmlns="http://www.w3.org/1999/xhtml"
2       xml:lang="en" lang="en">
3   <head
4       <meta http-equiv="Content-Type"
5           content="text/html; charset=utf-8" />
6       <title>simple</title>
7   </head>
8   <body>
9   <h1>A simple web page</h1>
10  <ul>
11      <li>List item one</li>
12      <li>List item two</li>
13  </ul>
14  <h2><a href="http://bil.omu.edu.tr">CENG</a></h2>
15  </body>
16  </html>
```

örnek

gösterim



→ HTML ile yazılmış tagle

→ hiyerarşiye dikkat. içiçelikler seviyelere karşılık geliyor

```
1  <body>
2      <ul>
3          <li>...</li>
```

5.3 sözlük ve tanımlar

düğüm (node): ağacın temel parçası. ismi var. "anahtar" olarak adlandırırız. ekstra bilgi ("payload") içerir. bir çok algoritma için payload önemsizdir.

dal (edge): iki düğümü birbirine bağlar. ilişkiyi gösterir.

kök (root): kendisine girdi olmayan tek düğüm. dosya sistemindeki /

yol (path): dalla bağlanan sıralı düğümler listesi. /etc/init.d/gdm

çocuk (children): aynı ebeveyne sahip düğümler. head:meta, title

ebeveyn (parent): çıktı üreten düğümler.

kardeş düğümler (sibling nodes): çocuklar kardeştir.

altağaç (subtree): özyineleme.

yaprak düğüm (leaf node): çocuğu olmayan düğüm

seviye (level): köke olan uzaklık. n=0 kök düğüm.

yükseklik (height): en büyük seviye. h=n_max HTML'de yükseklik= 3

tanım 1: ağaç

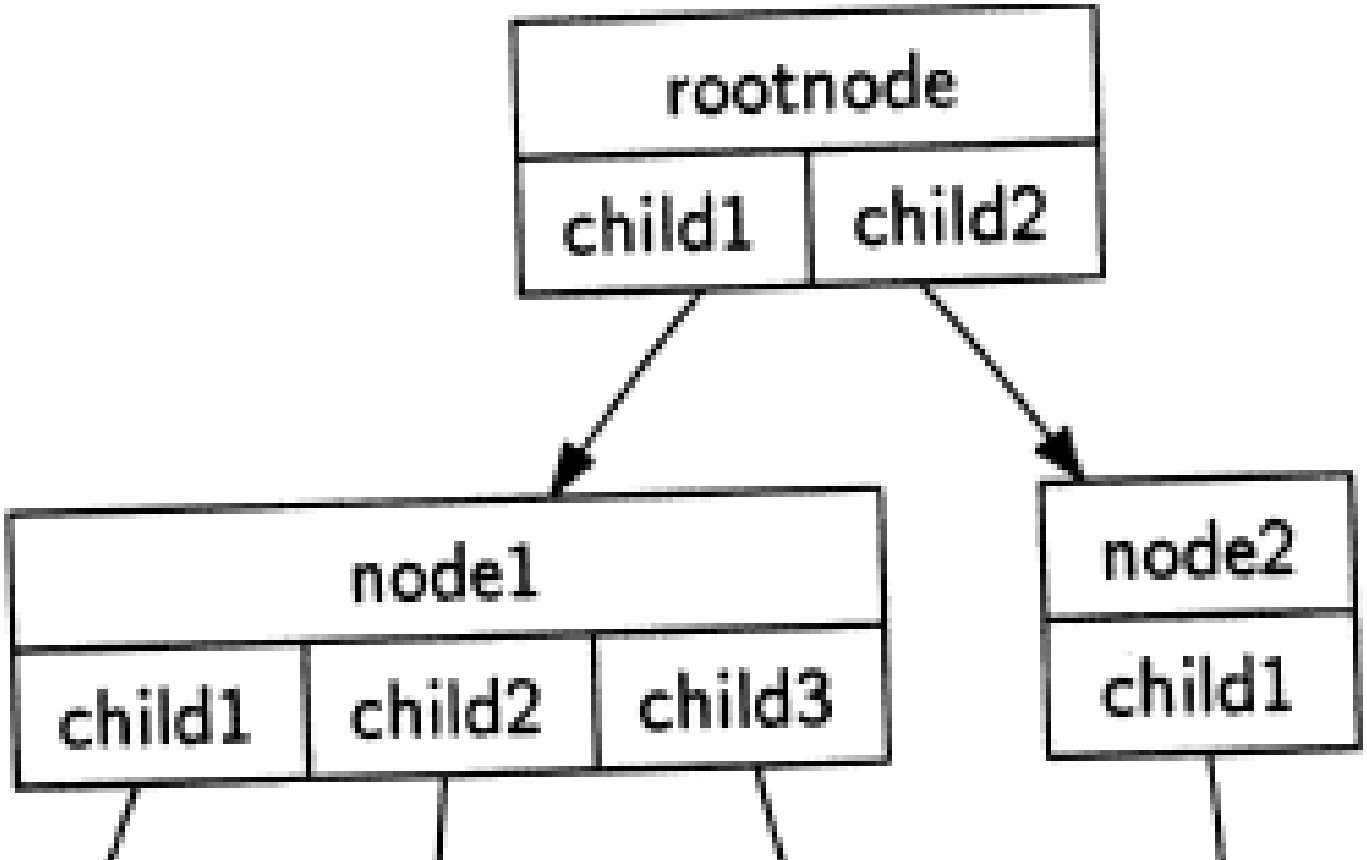
ağaç: düğüm kümesinden ve bunları birbirine bağlayan dallar kümesinden oluşur.

özellikleri:

→ ağacın ilk düğümü, köktür

→ kök hariç her düğümün ebeveyni vardır

→ kökten her bir düğüme yalnız bir yol vardır



tanım 1: ikil ağaç

ikil ağaç: her bir düğüm en fazla iki çocuğa sahip

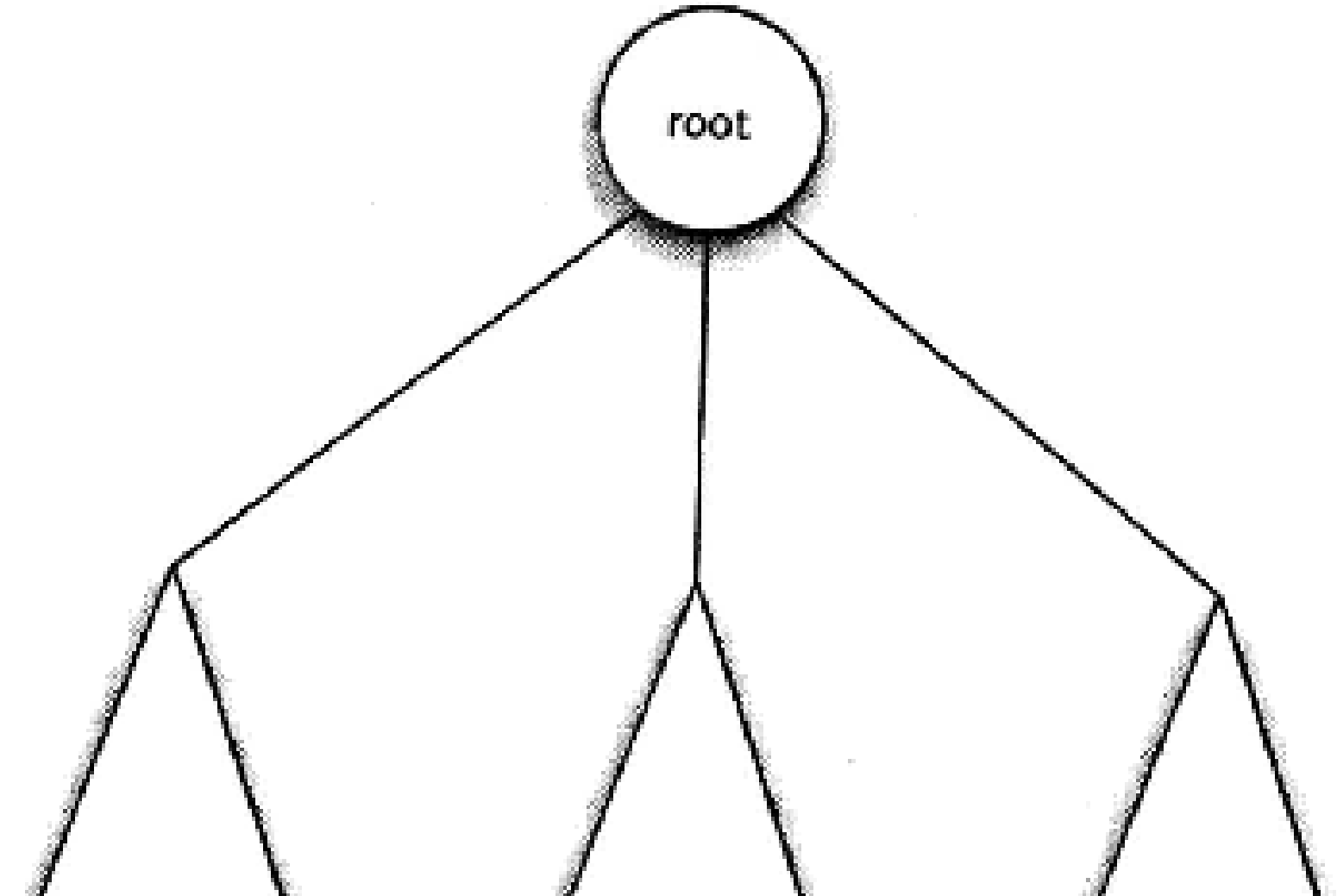
tanım 2: altağaç

ağaç, özyineleme

→ ya boş ya da bir kök içeren ve

→ sıfır veya daha fazla altağaçtan oluşur.

→ her bir ağacın kökü, ebeveynine bir dalla bağlıdır



5.4 gerçekleme

BinaryTree(): boş ağaç örneği

getLeftChild(): sol altağaç

getRightChild(): sağ altağaç

setRootVal(val): düğüme val ataması yapar

getRootVal(): düğümün değerini döndür

insertLeft(val): sol çocuğu ekle

insertRight(val): sağ çocuğu ekle

5.4.1 listelerin listesi temsili

genel yapı (özyinelilik),

```
Agac = [kok,
            [sol alt Agac],
            [sag alt Agac]
        ]
```

→ özyinelilik

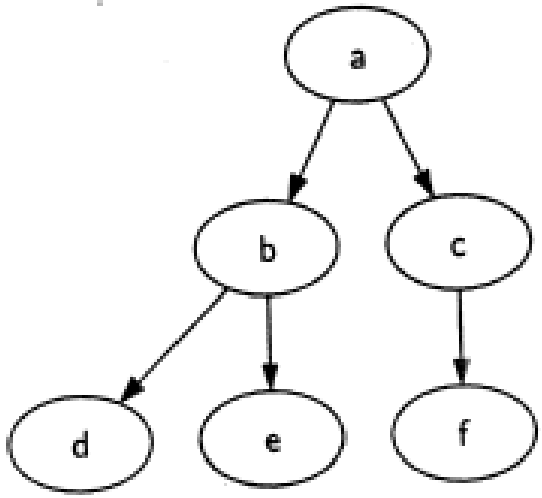
→ listenin ilk elemanı: Agac[0] $\Longrightarrow$ *kök*

→ listenin ikinci elemanı: Agac[1] $\Longrightarrow$ *solaltağaç*

→ listenin üçüncü elemanı: Agac[2] $\Longrightarrow$ *sağaltağaç*

örnek

basit bir ağaç



(a) A small tree

```
myTree = ['a',    #root
          ['b',   #left subtree
           ['d' [], []],
           ['e' [], []] ],
          ['c',   #right subtree
           ['f' [], []],
           [] ]
         ]
```

(b) The list representation of the tree

Figure 5.6: Representing a Tree As a List of Lists

kabuk

PB:5202

```
>>> myTree = ['a',  # kok
...             ['b', # sol altagac
...               ['d', [], []],
...               ['e', [], [] ],
...             ['c', # sag altagac
...               ['f', [], []],
...               [] ]
...           ]
>>> myTree
['a', ['b', ['d', [], []], ['e', [], []]], ['c', ['f', [], []], []]]
>>> kok = myTree[0]
>>> solAltAgac = myTree[1]
>>> sagAltAgac = myTree[2]
>>> kok
'a'
>>> solAltAgac
['b', ['d', [], []], ['e', [], []]]
>>> sagAltAgac
['c', ['f', [], []], []]
>>> solAltAgac[0] # kok
'b'
>>> solAltAgac[1] # sol alt agac
['d', [], []]
>>> solAltAgac[2] # sag alt agac
```

yardımcı işlevler

ikil ağaç oluştur

```
1       def BinaryTree(r):
2           return [r, [], []]
```

liste işlevlerini hatırlayalım

liste işlevleri

Method Name	Use	Explanation
append	`alist.append(item)`	Adds a new item to the end of a list
insert	`alist.insert(i,item)`	Inserts an item at the ith position in a list
pop	`alist.pop()`	Removes and returns the last item in a list
pop	`alist.pop(i)`	Removes and returns the ith item in a list
sort	`alist.sort()`	Modifies a list to be sorted
reverse	`alist.reverse()`	Modifies a list to be in reverse order
del	`del alist[i]`	Deletes the item in the ith position
index	`alist.index(item)`	Returns the index of the first occurrence of `item`
count	`alist.count(item)`	Returns the number of occurrences of `item`
remove	`alist.remove(item)`	Removes the first occurrence of `item`

Table 1.2: Methods Provided by Lists in Python

altağacı ekle

sol altağacı ekle

```
1       def insertLeft(root,newBranch):
2           t = root.pop(1)
3           if len(t) > 1:
4               root.insert(1,[newBranch,t,[]])
5           else:
6               root.insert(1,[newBranch, [], []])
7           return root
```

sağ altağacı ekle

```
def insertRight(root,newBra
    t = root.pop(2)
    if len(t) > 1:
        root.insert(2,[newB
    else:
        root.insert(2,[newB
    return root
```

→ yeni eklenen dal üste kalır

örnek çalışma

PB:5203

```
>>> from listing_5_1 import *
>>> from listing_5_2 import *
>>> from listing_5_3 import *
>>> myTree = BinaryTree('a')
>>> insertLeft(myTree, 'd')
['a', ['d', [], []], []]
>>> insertLeft(myTree, 'b')
['a', ['b', ['d', [], []], []], []]
>>> insertRight(myTree, 'f')
['a', ['b', ['d', [], []], []], ['f', [], []]]
>>> insertRight(myTree, 'c')
['a', ['b', ['d', [], []], []], ['c', [], ['f', [], []]]]
```

→ Fig 5.6'yı elde etmek için yeterli mi?

→ nasıl çağırdığımıza dikkat!

→ elde edilen ağacı çizin

get/set işlevleri

get/set işlevleri

```
1       def getRootVal(root):
2           return root[0]
3
4       def setRootVal(root,newVal):
5           root[0] = newVal
6
7       def getLeftChild(root):
8           return root[1]
9
10      def getRightChild(root):
11          return root[2]
```