Modul Praktikum
Struktur Data

Versi 16.03

Struktur Data Linier
Stack & Queue,
Deque & List Berkait
Rekursi, Pencarian dan Pengurutan
Binary Tree

Disusun oleh: Husni, MT.

Program Studi Teknik Informatika
Fakultas Teknik
Universitas Trunojoyo Madura
2016

Daftar Isi

Kata Pengantar

Alhamdulillah, modul praktikum Struktur Data edisi baru 16.03 ini dapat diselesaikan sesuai dengan jadwal pelaksanaan praktikum yang ditetapkan. Modul ini berbeda 100% dengan modul praktikum yang digunakan tahun sebelumnya (2015). Pada tahun ini, praktikum Struktur Data menggunakan bahasa Pemrograman Python, begitu pula pembahasan dalam modul ini, tepatnya Python versi 3 (terakhir 3.5.1).

Modul ini dibagi ke dalam 5 aktifitas lab yang direncanakan dapat dituntaskan dalam satu semester Praktikum. Walaupun jumlah pertemuannya sedikit, namun kandungannya sudah sangat mewakili konsep Struktur Data yang dibahas di kuliah, termasuk praktik tentang struktur data linier, Stack, Antrian (Queue), Deque (antrian dua pintu), List, Linked-list (list bersambungan), Tree (pohon biner), fungsi rekursif, Searching (pencarian) dan Sorting (pengurutan). Satu-satunya jenis struktur data yang tidak dijelaskan dalam modul ini adalah Graf (Graph). Graf merupakan materi terakhir dalam kuliah Struktur Data dan dapat dianggap sebagai topik paling berat. Dua alasan tersebut menyebabkan graf tidak dimasukkan dalam modul ini, mungkin dapat berubah di tahun depan.

Referensi utama dalam pembuatan modul ini adalah **Problem Solving with Algorithms and Data Structures** yang ditulis oleh Brad Miller dan David Ranum (dapat didownload di *interactivepython.org/runestone/static/pythonds/index.html*). Referensi lain yang sangat mendukung adalah buku **Data Structures and Algorithms in Python** karya Michael T. Goodrich, Roberto Tamassia dan Michael H. Goldwasser yang merupakan rujukan utama kuliah Struktur Data saat ini. Beberapa tulisan di Wikipedia juga turut melengkapi modul ini.

Kami mengucapkan terimakasih kepada Program Studi Teknik Informatika Universitas Trunojoyo Madura yang telah mempercayakan kami menata kembali materi Praktikum Struktur Data mengikuti tren bahasa pemrograman terkini. Terimakasih pula untuk para Asisten dan Laboran yang selalu berkoordinasi dalam penyelesaian modul ini. Kami menunggu dan berterimakasih atas semua feedback dari siapapun yang membaca modul ini, untuk menghasilkan modul yang lebih baik nantinya.

Semoga modul ini dapat memandu para Asisten Praktikum dan Praktikan dalam melaksanakan kegiatan Praktikum Struktur Data menggunakan Python.

Bangkalan, Maret 2016

Lab 1: Python: List, String, Tuple, Set, Dictionary dan Class

1.0 Obyektif

- Mereview ide-ide informatika, pemrograman dan penyelesaian masalah
- Memahami abstraksi dan peranannya dalam proses penyelesaian masalah
- Memahami dan mengimplementasikan *notion* dari tipe data abstrak
- Mereview bahasa pemrograman python.

1.1 List

Selain kelas numerik dan boolean, Python mempunyai sejumlah kelas koleksi built-in yang sangat powerful. Lists, strings dan tuples adalah koleksi terurut (*ordered*) yang sangat mirip secara struktur tetapi mempunyai perbedaan spesifik yang harus dipahami agar dapat digunakan dengan tepat. Sets dan dictionaries adalah koleksi tak-terurut (*unordered*).

List (daftar) adalah suatu koleksi berurut beranggotakan nol atau lebih rujukan ke obyek data Python. List ditulis sebagai nilai-nilai terpisahkan koma di dalam kurung siku. List kosong diwakili oleh [ ]. List bersifat heterogen, artinya obyek-obyek data di dalamnya tidak harus berasal dari kelas yang sama dan koleksi tersebut dapat berikan ke suatu variabel.

Coba tuliskan 3 baris instruksi ini di dalam modus interaktif (menggunakan software IDLE) dan perhatikan hasilnya:

```
>>>[1,3,True,6.5]
>>>myList = [1,3,True,6.5]
>>>myList
```

Tabel 1.1 Operasi terhadap Sequence di dalam Python

Nama Operasi	Operator	Penjelasan
Indexing	[]	Mengakses suatu elemen sequence
Concatenation	+	Menggabungkan sequence berama-sama
Repetition	*	Menggabungkan sejumlah berulang kali
Membership	In	Apakah suatu item adalah di dalam sequence
Length	Len	Jumlah item di dalam sequence
Slicing	[:]	Mengekstrak bagian dari suatu sequence

Coba jalankan instruksi-instruksi berikut pada modus Interaktif (IDLE):

```
>>>myList = [1,2,3,4]
>>>A = [myList]*3
```

```
>>>print(A)
>>>myList[2]=45
>>>print(A)
```

Tabel 1.2 Metode-metode yang disediakan oleh List di dalam Python

Nama Metode	Cara Penggunaan	Penjelasan
append	alist.append(item)	Menambahkan suatu item baru ke akhir list
insert	alist.insert(i,item)	Menyisipkan suatu item ke dalam list pada posisi ke-i
pop	alist.pop()	Menghapus & mengembalikan item terakhir dari dalam list
pop	alist.pop(i)	Menghapus dan mengembalikan item ke-i dari dalam list
sort	alist.sort()	Mengubah suatu list agar terurut
reverse	alist.reverse()	Mengubah suatu list dalam urutan terbalik
del	del alist[i]	Menghapus item pada posisi ke-i
index	alist.index(item)	Mengembalikan index dari kemunculan pertama dari item
count	alist.count(item)	Mengembalikan jumlah kehadiran dari item
remove	alist.remove(item)	Menghapus kemunculan pertama dari item

Berbagai operasi di atas dapat dilihat percobaannya di bawah ini:

```
>>>myList = [1024, 3, True, 6.5]
>>>myList.append(False)
>>>print(myList)
>>>myList.insert(2,4.5)
>>>print(myList)
>>>print(myList.pop())
>>>print(myList)

>>>print(myList.pop(1))
>>>print(myList)
>>>myList.pop(2)
>>>print(myList)
>>>myList.sort()
>>>print(myList)
>>>myList.reverse()
>>>print(myList)

>>>print(myList.count(6.5))
>>>print(myList.index(4.5))
>>>myList.remove(6.5)
>>>print(myList)
>>>del myList[0]
>>>print(myList)
```

1.2 String

String adalah koleksi sequential dari nol atau lebih huruf, bilangan dan simbol-simbol lain. Huruf-huruf, bilangan dan simbol-simbol lain tersebut dinamakan sebagai karakter. Nilai string literal dibedakan dari pengenal (identifiers) dengan menggunakan tanda quotation (single atau double). Cobalah 5 baris di bawah ini dan perhatikan hasilnya:

```
>>>"Susantoso"
>>>myName="Susantoso"
>>>myName[3]
>>>myName*2
>>>len(myName)
```

Berikut ini adalah contoh pemanfaatan metode yang tersedia dalam kelas Strings:

```
>>>myName
>>>myName.upper()
>>>myName.center(10)
>>>myName.find('v')
>>>myName.split('v')
```

Tabel 1.3 Metode bagi Strings di dalam Python

Nama Metode	Cara Penggunaan	Penjelasan
center	astring.center(w)	Mengembalikan string di tengah dalam field berukuran w
count	astring.count(item)	Mengembalikan jumlah kehadiran item dalam string
ljust	astring.ljust(w)	Mengembalikan string left-justified dalam field berukuran w
lower	astring.lower()	Mengembalikan string dalam huruf kecil
rjust	astring.rjust(w)	Mengembalikan string right-justified dalam field ukuran w
find	astring.find(item)	Mengembalikan index dari kemuculan pertama dari item
split	astring.split(schar)	Memecah string menjadi substring-substring pada schar

Perbedaan besar antara lists dan strings adalah bahwa lists dapat dimodifikasi sedangkan string tidak. Ini dikenal sebagai **mutability**. Lists bersifat mutable; strings dikatakan immutable. Kita dapat mengubah item dalam list menggunakan indexing dan assignment, tetapi hal tersebut tidak berlakukan untuk string.

```
>>>myList
>>>myList[0]=2**10
>>>myList
>>>myName
>>>myName[0]='X'
```

1.3 Tuple

Tuples sangat mirip dengan list dalam hal keberagaman rangkaian data. Perbedaannya adalah tuple bersifat *immutable*, seperti string. Tuples ditulis sebagai nilai-nilai dipisahan koma di dalam kurung. Karena berbentuk sequences maka operasi-operasi di atas dapat diterapkan terhadap tuples.

```
>>>myTuple=(2,True,4.96)
>>>myTuple
>>>len(myTuple)
>>>myTuple[0]
>>>myTuple*3
>>>myTuple[0:2]
```

1.4 Set

Suatu **set** adalah koleksi tak berurut dari nol atau lebih obyek data Python yang *immutable*. Sets tidak membolehkan duplikasi dan nilai-nilai dipisahkan koma di dalam kurung kurawal. Himpunan kosong diwakili oleh set(). Set bersifat heterogen, dan koleksi ini dapat diberikan ke suatu variabel.

```
>>>{3,6,"cat",4.5,False}
>>>mySet={3,6,"cat",4.5,False}
>>>mySet
```

Set tidak bersifat sequential, tetapi masih dapat menggunanakan operasi-operasi di atas. Tabel 4 merangkum operasi-operasi tersebut.

Tabel 1.4 Operasi terhadap Set di dalam Python

Nama Operasi	Operator	Penjelasan
Membership	in	Keanggotaan Himpunan (set)
Length	len	Mengembalikan kardinalitas dari himpunan
|	aset | otherset	Mengembalikan himpunan baru dengan semua elemen dari kedua set
&	aset & otherset	Mengembalikan set baru dengan hanya elemen-elemen yang hadir di kedua set
-	aset – otherset	Mengembalikan himpunan baru dengan semua item dari set pertama yang tidak ada di set kedua
<=	aset <= otherset	Menanyakan apakah semua elemen dari set pertama ada dalam set kedua?

```
>>>mySet
>>>len(mySet)
>>>False in mySet
>>>"dog" in mySet
```

```
>>>mySet
>>>yourSet={99,3,100}
>>>mySet.union(yourSet)
>>>mySet|yourSet
>>>mySet.intersection(yourSet)

>>>mySet&yourSet
>>>mySet.difference(yourSet)
>>>mySet-yourSet
>>>{3,100}.issubset(yourSet)
>>>{3,100}<=yourSet
>>>mySet.add("house")

>>>mySet
>>>mySet.remove(4.5)
>>>mySet
>>>mySet.pop()

>>>mySet
>>>mySet.clear()
>>>mySet
```

Tabel 1.5 Metode yang disediakan oleh Set di dalam Python

Nama Metode	Cara Penggunaan	Penjelasan
union	aset.union(otherset)	Mengembalikan himpunan baru dengan semua elemen dari kedua set
intersection	aset.intersection(otherset)	Mengembalikan himpunan baru dengan hanya elemen yang hadir di kedua set
difference	aset.difference(otherset)	Mengembalikan himpunan baru dengan semua item yang hadir dalam set pertama tetapi tidak hadir di dalam set kedua
issubset	aset.issubset(otherset)	Menanyakan apakah semua elemen dari set tertentu ada dalam set yang lain
add	aset.add(item)	Menambahkan item ke suatu set
remove	aset.remove(item)	Menghapus item dari suatu set
pop	aset.pop()	Menghapus elemen tertentu dari suatu set
clear	aset.clear()	Menghapus semua elemen dari suatu set

1.5 Dictionary

Koleksi Python terakhir dari struktur tak-berurut adalah **dictionary**. Dictionaries adalah koleksi pasangan item-item berasosiasi dimana setiap pasangan terdiri dari suatu key dan value. Pasangan key-value ini ditulis seabagai key:value. Dictionaries ditulis dipisahkan koma dalam kurung kurawal.

```
>>>capitals={'Iowa':'DesMoines','Wisconsin':'Madison'}
>>>capitals
```

Manipulasi terhadap dictionary dilakukan dengan mengakses nilai melalui key-nya atau dengan menambahkan pasangan key-value berikutnya. Mirip dengan mengaksesan sequence, tetapi tidak menggunakan inex, melainkan harus menggunakan key.

Listing 1.1 Program Python yang memanfaatkan dictionary

```
capitals = {'Iowa':'DesMoines','Wisconsin':'Madison'}
print(capitals['Iowa'])

capitals['Utah']='SaltLakeCity'
print(capitals)

capitals['California']='Sacramento'
print(len(capitals))

for k in capitals:
  print(capitals[k]," is the capital of ", k)
```

Tabel 1.6 Operator yang disediakan oleh Dictionaries di dalam Python

Operator	Penggunaan	Penjelasan
[]	mydict[k]	Mengembalikan nilai berasosiasi dengan k, jika tidak maka muncul error
in	key in adict	Mengembalikan True jika key ada dalam Dictionary, jika tidak False
del	del adict[key]	Menghapus entry dari Dictionary

Tabel 1.7 Metode yang disediakan Dictionaries di dalam Python

Nama Metode	Penggunaan	Penjelasan
keys	adict.keys()	Mengembalikan keys dari dictionary dalam obyek dict_keys
values	adict.values()	Mengembalikan values dari dictionary dalam obyek dict_values
items	adict.items()	Mengembalikan pasangan key-value pairs dalam obyek dict_items
get	adict.get(k)	Mengembalikan value yang berasosiasi dengan k, jika tidak None
get	adict.get(k, alt)	Mengembalikan nilai yang berasosiasi dengan k, jika tidak alt

```
>>>phoneext={'saalimah':1410, 'hidayah':1137}
>>>phoneext
>>>phoneext.keys()
>>>list(phoneext.keys())

>>>phoneext.values()
>>>list(phoneext.values())
>>>phoneext.items()
```

```
>>>list(phoneext.items())

>>>phoneext.get("kent")
>>>phoneext.get("kent","NO ENTRY")
```

Listing 1.2 Fungsi Menghitung Akar Kuadrat. Tambahkan exception handling agar root atau n dapat menerima nilai nol.

```
def squareroot(n):
        root = n / 2                #tebakan awal akan berupa 1/2 of n
        for k in range(20):
                root = (1 / 2) * (root + (n / root))
        return root

>>>squareroot(9)
>>>squareroot(4563)
```

1.6 Struktur Data Kelas: Mengenal Pendekatan Berorientasi Obyek

Listing 1.3 Class Fraction. Metode-metode aritmatika dan relasional ditinggalkan sebagai latihan

```
def gcd(m,n):
  while m%n != 0:
    oldm = m
    oldn = n

    m = oldn
    n = oldm%oldn
  return n

class Fraction:
   def __init__(self,top,bottom):
     self.num = top
     self.den = bottom

   def __str__(self):
     return str(self.num)+"/"+str(self.den)

   def show(self):
     print(self.num,"/",self.den)

   def __add__(self,otherfraction):
     newnum = self.num*otherfraction.den + \
              self.den*otherfraction.num
     newden = self.den * otherfraction.den
     common = gcd(newnum,newden)
     return Fraction(newnum//common,newden//common)
   def __eq__(self, other):
```

```
        firstnum = self.num * other.den
        secondnum = other.num * self.den

        return firstnum == secondnum

x = Fraction(1,2)
y = Fraction(2,3)
print(x+y)
print(x == y)
```

1.8 Rangkuman

- Informatika adalah kajian mengenai penyelesaian masalah berbasis komputer.
- Informatika menggunakan abstraksi sebagai perangkat untuk merepresentasikan proses dan data.
- Tipe data abstrak memungkinkan pemrogram mengelola kompleksitas dari suatu domain masalah dengan menyembunyikan detail dari datanya.
- Python adalah bahasa yang powerful, mudah digunakan dan object-oriented.
- Lists, tuples dan strings adalah koleksi sequential Python yang sudah built-in.
- Dictionaries dan sets adalah koleksi data non-sequential.
- Classes memungkinkan programmer mengimplementasikan tipe data abstrak.
- Programmer dapat meng-override metode-metode standard selain dapat membuat metode baru sendiri.
- Classes dapat diorganisasikan dalam bentuk hirarki.
- Konstruktor dari class akan selalu mengeksekusi konstruktor dari induknya sebelum menjalankan proses di dalam dirinya sendiri.

1.9 Tugas Pendahuluan

1. Implementasikan metode sederhana `getNum` dan `getDen` yang akan mengembalikan pembilang (numerator) dan penyebut (denominator) dari suatu pecahan (fraction).

1.10 Tugas Lanjutan

2. Implementasikan operator aritmatika sederhana (`__sub__`, `__mul__`, dan `__truediv__`).
3. Implementasikan operator relasional yang diperlukan: (`__gt__`, `__ge__`, `__lt__`, `__le__`, dan `__ne__`)
4. Modifikasi konstruktor untuk kelas fraction sehingga mampu memeriksa untuk memastikan bahwa pembilang dan penyebut keduanya bertipe integers. Jika salah satunya bukan integer, maka konstruktor akan memunculkan suatu exception.

1.11 Latihan Mandiri

5. Dalam definisi fraksi, kita menganggap bahwa pecahan negatif mempunyai pembilang negatif dan penyebut positif. Menggunakan suatu penyebut negatif dapat mengakibatkan beberapa operator relasi memberikan hasil tidak benar. Secara

umum, ini merupakan konstrain yang tidak penting. Lakukan perubahan terhadap konstruktor agar memungkinkan pengguna melewatkan suatu penyebut negatif sehingga semua operator tetap berlanjut bekerja dengan benar.

6. Telitilah metode __radd__ . Bagaimana perbedaannya dengan __add__? Kapan digunakan? Implementasikan metode__radd__.
7. Ulangi pertanyaan terakhir tetapi kali ini untuk metode __iadd__.
8. Telitilah metode __repr__ . Apa bedanya dengan __str__? Kapan digunakan? Implementasikan __repr__.
9. Rancanglah suatu kelas untuk merepresentasikan suatu permainan kartu. Rancanglah kelas unutk mereprentasikan kumpulan kartu. Menggunakan dua kelas ini, implementasikan suatu game kartu favorit anda.
10. Temukan permainan sudoku di majalah atau situs web. Tulislah program untuk menyelesaikan masalah puzzle tersebut.

Lab 2: Stack & Queue

2.0 Obyektif

- Memahami tipe data abstrak (Abstract Data Type, ADT) stack dan queue.
- Mampu mengimplementasikan ADTs stack dan queue menggunakan list dari Python.
- Memahami format ekspresi prefix, infix dan postfix dan menggunakan stacks untuk mengevaluasi ekspresi postfix, serta untuk mengkonversi ekspresi infix kepada postfix.
- Menggunakan queues untuk simulasi *timing* (pengaturan waktu) sederhana.
- Mampu mengenali properti-properti masalah dimana stacks dan queues merupakan struktur data yang tepat untuk digunakan.

2.1 Stack

Stacks, queues, deques dan lists merupakan contoh koleksi data yang item-itemmnya terurut bergantung pada bagaimana item tersebut ditambahkan atau dihapus. Begitu suatu item ditambahkan, ia mendiami posisi relatif terhadap elemen lain yang hadir sebelum dan setelahnya. Koleksi demikian dinamakan struktu data linier. Struktur linier mempunyai dua ujung, kadang dinamakan "kiri" dan "kanan", "depan" dan "belakang", juga "top" dan "base."

S**tack** (disebut pula "push-down stack") adalah koleksi berurut dari item-item dimana penambahan item baru dan penghapusan item yang telah ada selalu terjadi di ujung yang sama. Ujung ini dinamakan sebagai "top." Ujung berlawanan dari top dikenal sebagai "base." Stack diurutkan mengikuti konsep LIFO (*last in first out*). Berikut ini adalah beberapa operasi terhadap stack:

- **stack()** membuat suatu stack baru yang kosong. Tidak memerlukan parameter dan mengembalikan suatu stack kosong.
- **push(item)** menambahkan suatu item baru ke atas (top) dari stack. Perlu item dan tidak mengembalikan apapun.
- **pop()** menghapus item teratas dari stack. Tidak perlu parameter dan mengembalikan item. Stack berubah.
- **peek()** mengembalikan top item dari stack tetapi tidak menghapusnya. Tidak memerlukan parameter dan stack tidak berubah.
- **isEmpty()** memeriksa apakah stack dalam keadaan kosong. Tidak memerlukan parameter dan mengembalikan nilai boolean.
- **size()** mengembalikan jumlah item di dalam stack. Tidak memerlukan parameter dan mengembalikan suatu integer.

Sebagai contoh, jika `s` adalah suatu stack yang baru dibuat, tabel 2.1 memperlihatkan hasil serangkaian operasi terhadap stack.

Listing 2.1 Implementasi kelas stack dengan list. Ujung dari list dianggap memegang elemen teratas dari stack. Operasi **push** akan menambahkan item baru ke ujung list, operasi **pop** akan mengubah ujung yang sama tersebut.

```
class Stack:
    def __init__(self):
        self.items = []

    def isEmpty(self):
        return self.items == []

    def push(self, item):
        self.items.append(item)

    def pop(self):
        return self.items.pop()

    def peek(self):
        return self.items[len(self.items)-1]

    def size(self):
        return len(self.items)
```

Tabel 2.1 Contoh Operasi pada Stack

Operasi Stack	Isi Stack	Nilai Kembalian
s.isEmpty()	[]	True
s.push(4)	[4]	
s.push('dog')	[4,'dog']	
s.peek()	[4,'dog']	'dog'
s.push(True)	[4,'dog',True]	
s.size()	[4,'dog',True]	3
s.isEmpty()	[4,'dog',True]	False
s.push(8.4)	[4,'dog',True,8.4]	
s.pop()	[4,'dog',True]	8.4
s.pop()	[4,'dog']	True
s.size()	[4,'dog']	2

Listing 2.2 Contoh pemanfaatan kelas Stack yang ada di dalam modul pythonds:

```
from pythonds.basic.stack import Stack

s=Stack()

print(s.isEmpty())

s.push(4)
s.push('dog')
print(s.peek())

s.push(True)
print(s.size())
print(s.isEmpty())
```

```
s.push(8.4)
print(s.pop())
print(s.pop())
print(s.size())
```

Listing 2.3 Implementasi stack dengan list dimana top menunjukkan awal dari list, bukan ujungnya. Metode pop dan append sebelumnya tidak dapat digunakan lagi. Posisi indeks 0 (item pertama dalam list) dikenakan operasi pop dan push.

```
class Stack:

        def __init__(self):
                self.items = []

        def is_empty(self):
                return self.items == []

        def push(self, item):
                self.items.insert(0, item)

        def pop(self):
                return self.items.pop(0)

        def peek(self):
                return self.items[0]

        def size(self):
                return len(self.items)

s = Stack()
s.push('hello')
s.push('true')
print(s.pop())
```

Listing 2.4 Memeriksa keseimbangan jumlah tanda kurung. Algoritma dimulai dengan suatu stack kosong, memroses string berisi kurung dari kiri ke kanan. Jika suatu simbol adalah kurung buka, push ke stack sebagai simbol bahwa simbol tutup yang terkait harus muncul nantinya. Jika, pada sisi lain, suatu simbol adalah kurung tutup, pop stacknya. Selama masih mungkin melakukan pop terhadap stack untuk mencocokkan setiap simbol tutup, kurung tetap seimbang. Jika pada suatu waktu tidak ada simbol buka pada stack yang cocok dengan simbol tutup, string tersebut dikatakan tidak seimbang. Pada akhir string, ketika semua simbol telah diproses, stack akan kosong.

```
from pythonds.basic.stack import Stack

def parChecker(symbolString):
   s = Stack()
   balanced = True
   index = 0
```

```
    while index < len(symbolString) and balanced:
        symbol = symbolString[index]
        if symbol == "(":
            s.push(symbol)
        else:
            if s.isEmpty():
                balanced = False
            else:
                s.pop()

        index = index + 1

    if balanced and s.isEmpty():
        return True
    else:
        return False

print(parChecker('((()))'))
print(parChecker('(()'))
```

Listing 2.5 Konversi Bilangan Desimal ke Biner menggunakan algoritma bagi 2.

```
from pythonds.basic.stack import Stack

def divideBy2(decNumber):
    remstack = Stack()

    while decNumber > 0:
        rem = decNumber % 2
        remstack.push(rem)
        decNumber = decNumber // 2

    binString = ""
    while not remstack.isEmpty():
        binString = binString + str(remstack.pop())

    return binString

print(divideBy2(42))
```

Algoritma konversi biner dapat dieksten untuk mengerjakan konversi ke suatu basis. Bilangan desimal (233) dikonversi menjadi (351) oktal dan (E9) hexadecimal dapat oleh langkah-langkah berikut:

$3 * 8^2 + 5 * 8^1 + 1 \times 8^0$

Dan

$14 * 16^1 + 9 * 16^0$

Listing 2.6 Konversi suatu bilangan desimal ke basis bilangan tertentu

```
from pythonds.basic.stack import Stack

def baseConverter(decNumber,base):
   digits = "0123456789ABCDEF"

   remstack = Stack()

   while decNumber > 0:
      rem = decNumber % base
      remstack.push(rem)
      decNumber = decNumber // base

   newString = ""
   while not remstack.isEmpty():
      newString = newString + digits[remstack.pop()]

   return newString

print(baseConverter(25,2))
print(baseConverter(25,16))
```

Tabel 2.2 Contoh ekspresi infix, prefix dan postfix

Ekspresi Infix	Ekspresi Prefix	Ekspresi Postfix
A + B	+ A B	A B +
A + B * C	+ A * B C	A B C * +
(A + B) * C	* + A B C	A B + C *
A + B * C + D	+ + A * B C D	A B C * + D +
(A + B) * (C + D)	* + A B + C D	A B + C D + *
A * B + C * D	+ * A B * C D	A B * C D * +
A + B + C + D	+ + + A B C D	A B + C + D +

Listing 2.7 Konversi ekspresi Infix ke Postfix

```
from pythonds.basic.stack import Stack

def infixToPostfix(infixexpr):
   prec = {}
   prec["*"] = 3
   prec["/"] = 3
   prec["+"] = 2
   prec["-"] = 2
   prec["("] = 1
   opStack = Stack()
   postfixList = []
   tokenList = infixexpr.split()
```

```
    for token in tokenList:

        if token in "ABCDEFGHIJKLMNOPQRSTUVWXYZ" or token in "0123456789":
            postfixList.append(token)
        elif token == '(':
            opStack.push(token)
        elif token == ')':
            topToken = opStack.pop()

            while topToken != '(':
                postfixList.append(topToken)
                topToken = opStack.pop()

        else:
            while (not opStack.isEmpty()) and \
              (prec[opStack.peek()] >= prec[token]):
                postfixList.append(opStack.pop())
            opStack.push(token)

    while not opStack.isEmpty():
        postfixList.append(opStack.pop())
    return " ".join(postfixList)

print(infixToPostfix("A * B + C * D"))
print(infixToPostfix("( A + B ) * C - ( D - E ) * ( F + G )"))
```

Listing 2.8 Evaluasi Ekspresi Postfix

```
from pythonds.basic.stack import Stack

def postfixEval(postfixExpr):
    operandStack = Stack()
    tokenList = postfixExpr.split()

    for token in tokenList:
        if token in "0123456789":
            operandStack.push(int(token))
        else:
            operand2 = operandStack.pop()
            operand1 = operandStack.pop()
            result = doMath(token,operand1,operand2)
            operandStack.push(result)
    return operandStack.pop()

def doMath(op, op1, op2):
    if op == "*":
        return op1 * op2
    elif op == "/":
        return op1 / op2
    elif op == "+":
```

```
        return op1 + op2
    else:
        return op1 - op2

print(postfixEval('7 8 + 3 2 + /'))
```

2.2 Queue

Queue atau antrian adalah suatu koleksi item berurut dimana penambahan item baru terjadi di satu ujung bernama "ekor" (*rear*) dan penghapusan terjadi pada ujung lainnya yang dinamakan "kepala" (*front*). Queues menggunakan pengurutan FIFO. Berikut ini adalah beberapa operasi Queue:

- **queue()** membuat suatu antrian baru yang kosong. Tidak memerlukan parameter dan mengembalikan suatu antrian kosong.
- **enqueue(item)** menambahkan suatu item baru ke ujung saru antrian. Perlu item dan tidak mengembalikan sesuatu.
- **dequeue()** menghapus item depan dari antrian. Tidak memerlukan parameter dan mengembalikan itemnya. Antrian termodifikasi.
- **isEmpty()** menguji untuk melihat apakah antrian dalam keadaan kosong. Tidak memerlukan parameter dan mengembalian nilai boolean.
- **size()** mengembalikan jumlah item yang ada di dalam antrian. Tidak memerlukan parameter dan mengembalikan suatu integer.

Jika q adalah suatu queue yang baru dibuat (kosong) maka tabel 2.3 memperlihatkan serangkaian operasi terhadap queue tersebut.

Tabel 2.3 Contoh Operasi terhadap Queue

Operasi Antrian	Isi Antrian	Nilai Kembalian
q.isEmpty()	[]	True
q.enqueue(4)	[4]	
q.enqueue('dog')	['dog',4]	
q.enqueue(True)	[True,'dog',4]	
q.size()	[True,'dog',4]	3
q.isEmpty()	[True,'dog',4]	False
q.enqueue(8.4)	[8.4,True,'dog',4]	
q.dequeue()	[8.4,True,'dog']	4
q.dequeue()	[8.4,True]	'dog'
q.size()	[8.4,True]	2

Listing 2.9: Implementasi Queue (antrian) dimana ekor adalah pada posisi 0 dari list. Ini memungkinkan kita menggunakan fungsi insert untuk menambahkan elemen baru ke ekor dari antrian. Operasi pop dapat digunakan untuk menghapus elemen di depan (elemen terakhir dari list).

```
class Queue:

        def __init__(self):
                self.items=[]

        def isEmpty(self):
                return self.items==[]

        def enqueue(self,item):
                self.items.insert(0,item)

        def dequeue(self):
                return self.items.pop()

        def size(self):
                return len(self.items)

q=Queue()
q.enqueue(4)
q.enqueue('dog')

q.enqueue(True)
print(q.size())
```

Listing 2.10 Game untuk anak-anak “Kentang Panas” mensimulasikan antrian FIFO. Program menerima input suatu list nama dan konstanta bernama **num** yang digunakan sebagai counting (penghitungan). Program akan mengembalikan nama orang terakhir yang terisi setelah counting berulang sebanyak num.

```
from pythonds.basic.queue import Queue

def hotPotato(namelist, num):
   simqueue = Queue()

   for name in namelist:
      simqueue.enqueue(name)

   while simqueue.size() > 1:
      for i in range(num):
         simqueue.enqueue(simqueue.dequeue())

      simqueue.dequeue()

   return simqueue.dequeue()

print(hotPotato(["Bill","David","Susan","Jane","Kent","Brad"],7))
```

2.3 Simulasi: Antrian Mencetak dalam Jaringan

Langkah-langkah utama dari simulasi antrian mencetak ke suatu printer dari banyak komputer adalah
membuat suatu antrian (*queue*) tugas (*task*) mencetak. Setiap task akan diberikan suatu *timestamp* saat kedatangannya. Queue dimulai dalam keadaan kosong.

- Untuk setiap waktu (*currentSecond*): Apakah suatu tugas cetak akan dibuat? Jika Iya, maka tambahkan itu ke queue dengan currentSecond sebagai *timestamp*nya.
- Jika printer tidak sibuk dan jika suatu task sedang menunggu, hapus task berikutnya dari antrian cetak (print queue) dan serahkan task tersebut ke printer.
- Kurangkan timestamp dari currentSecond untuk menghitung waktu tunggu (*waiting time*) untuk task tersebut.
- Tambahkan *waiting time* untuk task tersebut ke suatu list untuk pemrosesan akan datang.
- Berdasarkan pada jumlah halaman dalam print task, tetapkan berapa waktu yang diperlukannya.
- Printer sekarang melakukan satu detik pencetakan jika diperlukan. Ia juga mengurangkan satu detik dari waktu yang diperlukan untuk tugas tersebut.
- Jika task tersebut telah selesai (*completed*), dengan kata lain waktu yang diperlukan telah mencapai nol, maka printer tidak sibuk lagi.
- Setelah simulasi selesai, hitung waktu tunggu rata-rata (*average waiting time*) dari daftar waktu tunggu yang dibangkitkan.

Simulasi ini dirancang dengan membuat kelas-kelas untuk tiga obyek nyata, yaitu: Printer, Task dan PrintQueue. Kelas Printer diperlukan untuk mencatat adanya kehadiran task (mencetak). Jika Ya, maka status printer menjadi “busy” dan waktu yang diperlukan dihitung dari jumlah halaman dalam task tersebut. Konstruktor memungkinkan perubahan terhadap *pages-per-minute*. Metode **tick** menurunkan internal timer dan mengatur printer ke status **idle** jika task sudah diselesaikan.

Listing 2.11 Kelas Printer

```
class Printer:
        def __init__(self,ppm):
                self.pagerate = ppm
                self.currentTask = None
                self.timeRemaining = 0

        def tick(self):
                if self.currentTask != None:
                        self.timeRemaining = self.timeRemaining-1

                        if self.timeRemaining <= 0:
                                self.currentTask = None

        def busy(self):
                if self.currentTask!=None:
                        return True
```

```
            else:
                return False

    defstartNext(self,newtask):
        self.currentTask = newtask
        self.timeRemaining = newtask.getPages()*60/self.pagerate
```

Kelas Task akan merepresentasikan suatu tugas printing. Saat task tersebut dibuat, suatu pembangkit bilangan acak menyediakan halaman dengan panjang 1 s.d 20. Kita memilih untuk menggunakan fungsi **randrange** dari modul random.

```
>>>importrandom
>>>random.randrange(1,21)
>>>random.randrange(1,21)
```

Setiap task perlu memelihara suatu timestamp yang digunakan untuk menghitung *waiting time*. Timestamp ini akan merepresentasikan waktu yang task yang dibuat dan diletakkan di dalam printer queue. Metode waitTime dapat digunakan kemudian untuk meretrieve jumlah waktu yang dihabiskan di dalam antrian semulai printing dimulai.

Listing 2.12 Kelas Task

```
import random

class Task:
    def __init__(self, time):
        self.timestamp = time
        self.pages = random.randrange(1,21)

    def getStamp(self):
        return self.timestamp

    def getPages(self):
        return self.pages

    def waitTime(self, currenttime):
        return currenttime - self.timestamp
```

Simulasi utama mengimplementasikan algoritma di atas. Obyek **printQueue** merupakan instance dari ADT queue yang ada. Fungsi boolean, **newPrintTask**, memutuskan apakah suatu printing task baru dibuatkan. Fungsi **randrange** digunakan untuk mengacak nilai 1 s.d 180. Print tasks tiba sekali setiap 180 detik. Fungsi **simulation** memungkinkan kita mengatur waktu total dan halaman per menit untuk printer tersebut.

Listing 2.13 Simulasi antrian pencetakan dokumen

```
from pythonds.basic.queue import Queue

import random
```

```
def simulation(numSeconds,pagesPerMinute):

        labprinter = Printer(pagesPerMinute)
        printQueue = Queue()
        waitingtimes = []

        for currentSecond in range (numSeconds):

                if newPrintTask():
                        task = Task(currentSecond)
                        printQueue.enqueue(task)

        if (not labprinter.busy()) and (not printQueue.isEmpty()):
                nexttask = printQueue.dequeue()
                waitingtimes.append(nexttask.waitTime(currentSecond))
                labprinter.startNext(nexttask)

        labprinter.tick()

        averageWait=sum(waitingtimes)/len(waitingtimes)
        print("Average Wait %6.2f secs %3d tasks remaining."%(averageWait,
        printQueue.size()))

def newPrintTask():
        num=random.randrange(1,181)

        if num==180:
                returnTrue
        else:
                return False

for i in range(10):
        simulation(3600,5)
```

Pada saat kita menjalankan simulasi, kita tidak perhatian dengan hasil yang berbeda setiap waktu. Ini karena sifat probabilistik dari bilangan random. Kita tertarik pada tren itu terjadi karena parameter simulation yang ditentukan.

Pertama, kita menjalankan simulation untuk periode 60 menit (3600 detik) menggunakan suatu *page rate* 5 halaman per menit. Sebagai tambahan, kita menjalankan 10 percobaan independen. Ingat bahwa karena simulation bekerja dengan bilangan random maka setiap run akan mengembalikan hasil berbeda.

```
>>>for i in range(10):
    simulation(3600,5)
```

Setelah menjalankan 10 trial kita dapat melihat bahwa waktu tunggu (*wait time*) rata-rata adalah 122.155 seconds. Dapat pula dilihat bahwa ada variasi besar dalam waktu bobot rata-rata dengan suatu rata-rata minimum 17.27 seconds dam maksimum 239.61 seconds. Juga terlihat bahwa hanya dalam dua kasus semua task tuntas lengkap.

Sekarang, kita akan mengatur page rate menjadi 10 pages per minute dan menjalankan 10 trials lagi, dengan suatu *page rate* lebih cepat diharapkan lebih banyak tugas akan diselesaikan dalam frame waktu satu jam.

```
>>>for i in range(10):
    simulation(3600,10)
```

Listing 2.14 Kode program lengkap dari proses yang dijelaskan di atas (3 kelas dan metode di dalamnya)

```
from pythonds.basic.queue import Queue

import random

class Printer:
   def __init__(self, ppm):
      self.pagerate = ppm
      self.currentTask = None
      self.timeRemaining = 0

   def tick(self):
      if self.currentTask != None:
         self.timeRemaining = self.timeRemaining - 1
         if self.timeRemaining <= 0:
            self.currentTask = None

   def busy(self):
      if self.currentTask != None:
         return True
      else:
         return False

   def startNext(self,newtask):
      self.currentTask = newtask
      self.timeRemaining = newtask.getPages() * 60/self.pagerate

class Task:
   def __init__(self,time):
      self.timestamp = time
      self.pages = random.randrange(1,21)

   def getStamp(self):
      return self.timestamp

   def getPages(self):
      return self.pages

   def waitTime(self, currenttime):
      return currenttime - self.timestamp
```

```
def simulation(numSeconds, pagesPerMinute):

   labprinter = Printer(pagesPerMinute)
   printQueue = Queue()
   waitingtimes = []

   for currentSecond in range(numSeconds):

     if newPrintTask():
        task = Task(currentSecond)
        printQueue.enqueue(task)

     if (not labprinter.busy()) and (not printQueue.isEmpty()):
       nexttask = printQueue.dequeue()
       waitingtimes.append( nexttask.waitTime(currentSecond))
       labprinter.startNext(nexttask)

     labprinter.tick()

   averageWait=sum(waitingtimes)/len(waitingtimes)
   print("Average Wait %6.2f secs %3d tasks remaining."%(averageWait,printQueue.size()))

def newPrintTask():
   num = random.randrange(1,181)
   if num == 180:
      return True
   else:
      return False

for i in range(10):
   simulation(3600,5)
```

2.4 Rangkuman

- Struktur data linier memelihara datanya dalam bentuk terurut.
- Stacks merupakan struktur data sederhana yang bekerja dengan skema LIFO, *last-in first-out*.
- Operasi fundamental terhadap stack adalah push, pop dan is_empty.
- Queues merupakan struktur data simpel yang bekerja dengan konsep FIFO, *first-in first-out*.
- Operasi fundamental bagi suatu queue mencakup enqueue, dequeue dan is_empty.
- Prefix, infix dan postfix adalah cara menuliskan ekspresi.
- Stacks sangat berdayaguna untuk perancangan algoritma untuk mengevaluasi dan menerjemahkan ekspresi.
- Stacks dapat menyediakan karakteristik pembalikan (*reversal*).
- Queues dapat membantu pembangunan simulasi pe-waktu-an (*timing*).

- Simulasi adalah pembangkit bilangan random (acak) untuk membuat situasi kehidupan nyata (real-life) dan memungkinkan kita untuk menjawab pertanya berjenis “what if” (bagaimana jika?)

2.5 Tugas Pendahuluan

1. Buatlah suatu fungsi bernama **revString(mystr)** yang menggunakan suatu stack untuk membalikkan karakter-karakter dalam suatu string. Buat juga suatu fungsi bernama **testEqual()** untuk memeriksa apakah dua string bernilai sama atau tidak. Template dari code yang akan dibuat seperti di bawah ini:

```
def testEqual(string1, string2):
        # tuliskan kode untuk membandingkan kesamaan dua string

def revString(mystr):
        # tuliskan kode untuk membalikkan karakter dalam string

testEqual(revString('apple'), 'elppa')
testEqual(revString('x'), 'x')
testEqual(revString('1234567890'), '0987654321')
```

2.6 Tugas Lanjutan

2. Implementasikan ADT Queue, menggunakan suatu list sehingga ekor dari antrian adalah pada ujung dari list.
3. Contoh lain dari masalah pencocokan tanda kurung datang dari hypertext markup language (HTML, halaman web dasar). Dalam HTML, tag-tag hadir dalam bentuk opening (buka) dan closing (tutup) dan harus seimbang untuk dengan tepat mendeskripsikan suatu dokumen web. Ini adalah contoh dokumen HTML yang simpel:

```
<html>
<head>
        <title>
                Contoh Halaman Web
        </title>
</head>

<body>
        <h1>Halo kawan, Saya belajar Python dan HTML</h1>
</body>
</html>
```

   Terlihat jelas bahwa suatu tag buka, misal <html> mempunyai pasangan tag tutup </html>. Tulislah suatu program Python yang mampu memeriksa kehadiran tag buka dan tutup secara benar di dalam dokumen HTML.

4. Lakukan perubahan terhadap algoritma dan kode program Python infix-to-postfix sehingga mampu menangani error.

2.7 Latihan Mandiri

5. Lakukan perubahan terhadap algoritma evaluasi postfix sehingga ia mampu menangani kemungkinan error.
6. Implementasikan suatu evaluator direct infix yang menggabungkan fungsionalitas algoritma dari konversi infix-to-postfix dan evaluasi postfix. Evaluator tersebut sebaiknyanya memroses token-token infix dari kiri ke kanan dan menggunakan dua stack, satu untuk operator dan satunya untuk operan, untuk mengerjakan evaluasi.
7. Lakukan perubahan dari evaluator infix sebelumnya menjadi suatu calculator.
8. Rancang dan implementrasikan suatu eksperimen untuk melakukan perbandingan benchmark dari dua implementasi queue. Apa yang dapat kita pelajari dari eksperimen ini?
9. Adalah mungkin untuk mengimplementasikan suatu antrian (queue) sehingga enqueue dan dequeue mempunyai kinerja rata-rata O(1). Pada kasus ini berarti bahwa sebagian besar waktu enqueue dan dequeue akan menjadi O(1) kecuali satu keadaan tertentu dimana dequeue akan berupa O(n).
10. Perhatikan situasi kehidupan nyata. Formulasikan suatu pertanyaan dan kemudian rancang suatu simulasi yang dapat membantu menjawabnya. Situasi yang mungkin termasuk:

    a. Posisi antrian mobil pada suatu tempat pencucian mobil (*car wash*)
    b. Antrian pelanggan pada kasir supermarket (*grocery store*)
    c. *Take-off* dan *landing* pesawat terbang pada suatu *run way* (di bandara)
    d. Seorang *teller* di bank

    Pastikan untuk menyatakan suatu asumsi bahwa anda membuat dan menyediakan suatu data probabilistik yang harus dipertimbangkan sebagai bagian dari *scenario*.

11. Lakukan perubahan terhadap simulasi permainan Hot Potato agar memungkinkan nilai counting terpilih secara acak sehingga setiap pass tidak dapat diprediksi.
12. Implementasikan suatu mesin pengurutan radix. Pengurutan radix untuk Interger basis 10 adalah suatu teknik pengurutan mekanis yang menggunakan sederetan keranjang, satu keranjang utama (*main*) dan 10 keranjang digit (0 s.d 9) . Setiap keranjang bertindak seperti suatu antrian dan memegang nilai-nilainya (terurut) sesuai dengan kedatangannnya. Algortima bermula dengan penempatan setiap bilangan dalam keranjang utama. Kemudian melihat setiap nilai digit demi digit. Nilai pertama dihapus dan ditempatkan ke dalam keranjang digit yang berkaitan dengan digit yang dilihat. Sebagai contoh, jika suatu digit yang dipertimbangkan/sedang dilihat, 534 ditempatkan dalam keranjang digit 4 dan 667 diletakkan di keranjang digit 7. Setelah semua nilai diletakkan di dalam keranjang dikit yang bersesuaian, nilai-nilai dikoleksi dari keranjang 0 s.d 9 dan diletakkan kembali ke dalam keranjang utama. Proses tersebut berlanjut dengan digit puluhan, ratusan, dan seterusnya. Setelah digit terakhir diproses, keranjang utama mengandung nilai-nilai yang terurut.

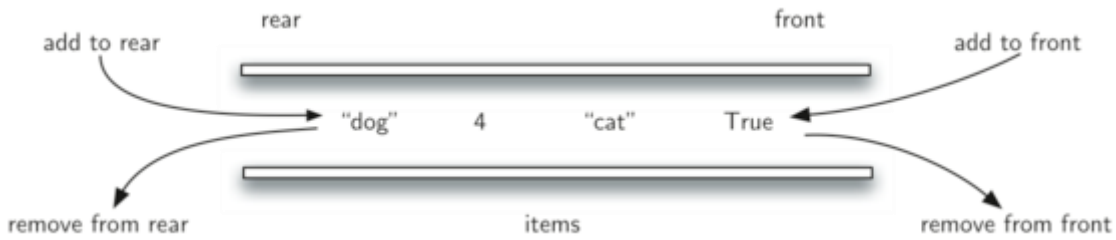
Lab 3: Deque & Linked-List

3.0 Obyektif

- Memahami ADT deque dan list.
- Mampu mengimplementasikan ADT deque dan list berkait menggunakan list dari Python.
- Mampu mengenali properti-properti masalah dimana deques merupakan struktur data yang tepat.
- Mampu mengimplementasikan ADT list sebagai suatu linked list menggunakan pola node dan referensi.
- Mampu membandingkan kinerja dari implementasi linked list dengan implementasi list dari Python.

3.1 Deque

D**eque (Deantrian)**, dikenal juga sebagai antrian berujung dua (*double-ended*), adalah suatu koleksi item terurut serupa dengan queue. Perbedaannya? Sifat tidak terikat untuk penambahan dan penghapusan item-item dari antrian. Item baru dapat ditambahkan ke depan atau belakang. Karena itu, item yang ada dapat dihapuskan dari salah satu ujung. Struktur linier hibrida ini menyediakan semua kapabilutas stack dan antrian dalam satu struktur data.



Gambar 3.1 Obyek Data Deque

Berikut ini adalah beberapa operasi yang dapat diberlakukan terhadap deque:

- **deque()** membuat suatu deque baru yang kosong. Tidak perlu parameter dan mengembalikan suatu deque kosong.
- **addFront(item)** menambahkan suatu item baru ke depan dari deque. Perlu item dan tidak mengembalikan apapun.
- **addRear(item)** menambahkan suatu item baru ke ekor dari deque. Perlu item dam tidak mengembalikan sesuatu.
- **removeFront()** menghapus item depan dari deque. Tidak perlu parameter dan mengembalikan item. Deque termodifikasi.
- **removeRear()** menghapus item ujung (ekor) dari deque. Tidak perlu parameter dan mengembalikan item. Deque berubah.
- **isEmpty()** menguji apakah deque dalam kondisi kosong. Tidak perlu parameter dan mengembalikan suatu nilai boolean.

- **size()** mengembalikan jumlah item dalam deque. Tidak perlu parameter dan mengembalikan suatu integer.

Jika d adalah deque yang baru dibuat dan kosong, tabel 3.1 memperlihatkan contoh operasi terhadap d. Isi dari ujung depan (*front*) diletakkan di kanan. Penting sekali untuk menjaga track dari depan dan belakang (*rear*) karena item-item dapat dipindahkan (keluar masuk koleksi) secara bebas yang mungkin sedikit membingungkan.

Tabel 3.1 Contoh operasi terhadap deque

Operasi Deque	Isi Deque	Nilai Kembalian
d.isEmpty()	[]	True
d.addRear(4)	[4]	
d.addRear('dog')	['dog',4,]	
d.addFront('cat')	['dog',4,'cat']	
d.addFront(True)	['dog',4,'cat',True]	
d.size()	['dog',4,'cat',True]	4
d.isEmpty()	['dog',4,'cat',True]	False
d.addRear(8.4)	[8.4,'dog',4,'cat',True]	
d.removeRear()	['dog',4,'cat',True]	8.4
d.removeFront()	['dog',4,'cat']	True

Listing 3.1 Implementasi ADT deque, ekor dari deque pada posisi 0 dari list

```
class Deque

def __init__(self):
      self.items = []

   def isEmpty(self):
      return self.items == []

   def addFront(self, item):
      self.items.append(item)

   def addRear(self, item):
      self.items.insert(0,item)

   def removeFront(self):
      return self.items.pop()

   def removeRear(self):
      return self.items.pop(0)

   def size(self):
      return len(self.items)

d = Deque()
print(d.isEmpty())
```

```
d.addRear(4)
d.addRear('dog')
d.addFront('cat')
d.addFront(True)
print(d.size())
print(d.isEmpty())

d.addRear(8.4)
print(d.removeRear())
print(d.removeFront())
```

Dalam removeFront digunakan metode pop untuk menghapus elemen terakhir dari list. Namun, dalam removeRear, metode pop(0) harus menghapus elemen pertama dari list. Metode insert digunakan dalam addRear karena metode append menganggap penambahan elemen baru ke ujung (akhir) list.

Listing 3.2 Palindrome-Checker menggunakan deque

```
from pythonds.basic.deque import Deque

def palchecker(aString):
   chardeque = Deque()

   for ch in aString:
      chardeque.addRear(ch)

   stillEqual = True

   while chardeque.size() > 1 and stillEqual:
      first = chardeque.removeFront()
      last = chardeque.removeRear()
      if first != last:
         stillEqual = False

   return stillEqual

print(palchecker("lsdkjfskf"))
print(palchecker("radar"))
```

3.2 List Lanjutan

Sejauh ini, kita sudah menggunakan Python lists untuk mengimplementasikan berbagai tipe data abstrak. List bersifat powerful namun simpel, mekanisme koleksi yang menyediakan *programmer* variasi operasi yang luas. Namun tidak semua bahasa pemrograman menyertakan suatu koleksi list. Pada kasus demikian, gagasan mengenai list harus dimplementasikan oleh programmer.

List adalah koleksi item dimana setiap item berposisi relatif terhadap yang lain. Ini disebut pula *unordered list*. List punya item pertama, kedua, kedtiga dan seterusnya. Dapat pula dirujuk awal list (item pertama) atau ujung list (item terakhir). Untuk penyederhanaan, anggap list tidak dapat mengandung item-item terduplikat (ganda).

Beberapa operasi list tak-berurut (*unordered*) adalah

- **List()** membuat suatu list baru yang kosong. Tidak memerlukan parameter dan mengembalikan suatu list kosong.
- **add(item)** menambahkan suatu item baru ke dalam list. Diperlukan item yang akan ditambahkan dan tidak mengembalikan apapun. Anggapan: item tersebut belum ada dalam list.
- **remove(item)** menghapus item dari dalam list. Diperlukan item dan akan mengubah list. Anggapan: item tersebut tersedia di dalam list.
- **search(item)** mencari item di dalam list. Perlu item dan mengembalikan suatu nilai boolean.
- **isEmpty()** menguji untuk melihat apakah list dalam keadaan kosong (*empty*). Tidak memerlukan parameter dan mengembalikan suatu nilai boolean.
- **size()** mengembalikan jumlah item di dalam list. Tidak memerlukan parametyer dan mengembalikan suatu integer.
- **append(item)** menambahkan item baru ke akhir (ujung) list dan menjadikannya sebagai item terakhir di dalam koleksi. Diperlukan item dan tidak mengembalikan apapun. Anggapan: item tersebut belum ada di dalam list.
- **index(item)** mengembalikan posisi dari item di dalam list. Diperlukan item dan mengembalikan index dari item tersebut. Anggapan: item terdapat di dalam list.
- **insert(pos, item)** menambahkan suatu item baru ke dalam list pada posisi pos. Diperlukan item dan tidak mengembalikan apapun. Anggapan: item belum terdapat di dalam list dan ada cukup item existing (telah ada di dalam list) sehingga dapat diketahui posisi pos.
- **pop()** menghapus dan mengembalikan item terakhir di dalam list. Tidak memerlukan parameter dan mengembalikan suatu item. Anggapan: list tidak kosong, setidaknya ada satu item.
- **pop(pos)** menghapus dan mengembalikan item pada posisi pos. Diperlukan posisi sebagai parameter dan mengembalikan suatu item. Anggapan: item ada dalam list.

Listing 3.2 Kelas Node dan UnorderedList yang mengimplementasikan suatu linked-list (list berkait atau bersambungan). Perlu ditentukan nilai data awal untuk node. Juga ada metode untuk mengakses dan mengubah datanya dan referensi berikutnya. Perhatikan konstruktor, metode add, search, remove dan bagaimana itu semua digunakan.

```
class Node:

  def __init__(self,initdata):
    self.data = initdata
    self.next = None

  def getData(self):
    return self.data
```

```
    def getNext(self):
        return self.next

    def setData(self,newdata):
        self.data = newdata

    def setNext(self,newnext):
        self.next = newnext

class UnorderedList:

    def __init__(self):
        self.head = None

    def isEmpty(self):
        return self.head == None

    def add(self,item):
        temp = Node(item)
        temp.setNext(self.head)
        self.head = temp

    def size(self):
        current = self.head
        count = 0
        while current != None:
            count = count + 1
            current = current.getNext()

        return count

    def search(self,item):
        current = self.head
        found = False

        while current != None and not found:
            if current.getData() == item:
                found = True
            else:
                current = current.getNext()

        return found

    def remove(self,item):
        current = self.head
        previous = None
        found = False
        while not found:
            if current.getData() == item:
                found = True
```

```
            else:
                previous = current
                current = current.getNext()

        if previous == None:
            self.head = current.getNext()
        else:
            previous.setNext(current.getNext())

mylist = UnorderedList()

mylist.add(31)
mylist.add(77)
mylist.add(17)
mylist.add(93)
mylist.add(26)
mylist.add(54)

print(mylist.size())
print(mylist.search(93))
print(mylist.search(100))

mylist.add(100)
print(mylist.search(100))
print(mylist.size())

mylist.remove(54)
print(mylist.size())
mylist.remove(93)
print(mylist.size())
mylist.remove(31)
print(mylist.size())
print(mylist.search(93))
```

Metode-metode lain diperlukan dalam mengelola list di atas, misalnya append, insert, index, dan pop. Jadikan itu sebagai latihan (dikerjakan secara mandiri).

Bagaimana dengan ADT List berurut (*ordered*)? Jika list integer di atas dibuat ordered list (urut *ascending*) maka dapat ditulis sebagai 17, 26, 31, 54, 77 dan 93. Karena 17 adalah item terkecil, maka ia menempati posisi pertama dalam list. Demikian juga, karena 93 adalah yang terbesar, ia menempati posisi terakhir.

Berikut ini adalah beberapa operasi yang dapat diberlakukan terhadap list berurut:

- **OrderedList()** membuat suatu ordered list baru yang masih kosong. Tidak memerlukan parameter dan mengembalikan suatu list kosong.
- **add(item)** menambahkan suatu item baru ke dalam list, urutan dipersiapkan. Memerlukan item sebagai parameter dan tidak mengembalikan apapun. Anggapan: item tidak ada di dalam list.

- **remove(item)** menghapus item dari list. Memerlukan item sebagai parameter dab akan memodifikasi list. Anggapan: item telah ada di dalam list.
- **search(item)** mencari item di dalam list. Perlu item sebagai parameter dan mengembalikan suatu nilai boolean.
- **isEmpty()** menguji apakah list dalam kondisi kosong. Tidak memerlukan parameter dan mengembalikan suatu nilai boolean.
- **size()** mengembalikan jumlah item dalam list. Tidak memerlukan parameter dan mengembalikan suatu integer.
- **index(item)** mengembalikan posisi dari item di dalam list. Memerlukan item sebagai parameter dan mengembalikan index. Anggapan: item ada di dalam list.
- **pop()** menghapus dan mengembalikan item terakhir dalam list. Tidak memerlukan parameter dan mengembalikan suatu item. Anggapan: list mempunyai setidaknya satu item.
- **pop(pos)** menghapus dan mengembalikan item pada posisi pos. Perlu posisi sebagai parameter dan mengembalikan item. Anggapan: item ada di dalam list.

Listing: Implementasi kelas Node dan OrderedList serta pemanfaatannya

```
class Node:

   def __init__ (self, initdata):
      self.data = initdata
      self.next = None

   def getData(self):
      return self.data

   def getNext(self):
      return self.next

   def setData(self, newdata):
      self.data = newdata

   def setNext(self, newnext):
      self.next = newnext

class OrderedList:

   def __init__(self):
      self.head = None

   def search(self, item):
      current = self.head
      found = False
      stop = False

      while current != None and not found and not stop:
         if current.getData() == item:
            found = True
```

```
            else:
                if current.getData() > item:
                    stop = True
                else:
                    current = current.getNext()

        return found

    def add(self, item):
        current = self.head
        previous = None
        stop = False
        while current != None and not stop:
            if current.getData() > item:
                stop = True
            else:
                previous = current
                current = current.getNext()

        temp = Node(item)
        if previous == None:
            temp.setNext(self.head)
            self.head = temp
        else:
            temp.setNext(current)
            previous.setNext(temp)

    def isEmpty(self):
        return self.head == None

    def size(self):
        current = self.head
        count = 0
        while current != None:
            count = count + 1
            current = current.getNext()

        return count

mylist = OrderedList()
mylist.add(31)
mylist.add(77)
mylist.add(17)
mylist.add(93)
mylist.add(26)
mylist.add(54)

print(mylist.size())
print(mylist.search(93))
print(mylist.search(100))
```

3.3 Rangkuman

- Deques adalah struktur data yang memungkinkan perilaku hibrid seperti stacks dan queues.
- Operasi fundamental bagi suatu deque adalah addFront, addRear, removeFront, removeRear dan isEmpty.
- Lists merupakan koleksi item-item dimana setiap item memegang suatu posisi relatif.
- Implementasi linked list memelihara *logical order* tanpa memerlukan persyaratan *physical storage*.
- Modifikasi terhadap head dari linked list merupakan kasus khusus.

3.4 Tugas Pendahuluan

1. Anda sudah melihat bagaimana deque digunakan untuk memeriksa palindrom tidaknya suatu String. Lakukan modifikasi terhadap program Python sebelumnya sehingga dapat memeriksa teks palindrom yang mengandung spasi, misalnya ”I PREFER PI” adalah string yang harus dianggap palindrom.

3.5 Tugas Lanjutan

1. Implementasikan suatu stack menggunakan linked lists.
2. Impmenetasi metode length (menghitung panjang), biasanya dengan menghitung jumlah elemen/node/item dalam list. Strategi alternatifnya adalah dengan menyimpan jumlah node di dalam list sebagai potongan data tambahan di dalam kepala (head) dari list tersebut. Lakukan perubahan terhadap kelas UnorderedList untuk memasukkan informasi ini dan tulis ulang (betulkan) metode length()-nya.
3. Implementasikan metode remove() sehingga dapat bekerja dengan tepat saat tidak ada item di dalam list (list dalam keadaan kosong).

3.6 Latihan Mandiri

4. Lakukan perubahan terhadap kelas List agar membolehkan duplikasi. Metode-metode mana saja yang akan terpengaruh dengan perubahan ini?
5. Implementasikan metode __str__ di dalam kelas UnorderedList. Seperti apa representasi String yang bagus untuk suatu list?
6. Implementasikan metode __str__ sehingga list ditampilkan dengan cara Python (di dalam kurung siku buka dan tutup).
7. Implementasikan operasi-operasi yang tersisi di dalam ADT UnorderedList (seperti append, index, pop dan insert).
8. Implementasikan suatu metode slice untuk kelas UnorderedList. Metode ini memerlukan dua parameter: start dan stop, dan mengembalikan suatu salinan dari list dimulai pada posisi start dan seterusnya tetapi tidak termasuk pada posisi stop.
9. Implementasikan operasi yang belum ada di dalam ADT OrderedList.
10. Perhatikan relasi antar alist Unordered dan Ordered. Apakah mungkin konsep inheritance digunakan untuk membangun implementasi yang lebih efisien? Implementasikan hirarki inheritance ini.

11. Implementasikan suatu queue menggunakan linked lists.
12. Implementasi suatu deque menggunakan linked lists.
13. Rancang dan implementasikan suatu eksperimen yang akan membandingkan kinerja list di Python (bawaan) dengan list yang diimplementasikan sebagai suatu linked list.
14. Rancang dan implementasikan suatu eksprerimen yang akan membandingkan kinerja dari Python list berbasis stack dan queue dengan implementasi linked list.
15. Implementasi linked list di atas dinamakan suatu *singly linked list* (list bersambung satu demi satu) karena setiap node mempunyai referensi tunggal ke node berikutnya di dalam rangkaian. Implementasi alternatifnya dikenal sebagai *doubly linked list*. Dalam implementasi ini, setiap node mempunyai suatu referensi ke node berikutnya (next node, disebut next) juga suatu referensi ke node sebelumnya (biasa dipanggil back). Referensi head juga mengandung dua referensi, satu menuju node pertama dalam linked list dan satu lagi ke node terakhir. Tuliskan kode implementasi ini dalam Python.

Lab 4: Rekursi, Searching & Sorting

4.0 Obyektif

- Memahami bahwa masalah kompleks sebagian dapat disederhanakan dengan solusi rekursif.
- Mempelajari bagaimana menformulasikan program secara rekursif.
- Memahami dan mengaplikasikan 3 hukum rekursi.
- Memahami rekursi sebagai suatu bentuk iterasi.
- Memahami formulasi rekursif dari suatu masalah.
- Memahami bagaimana rekursi diimplementasikan oleh suatu sistem komputer.
- Mampu menjelaskan dan mengimplementasikan pencarian sequential dan binary.
- Memahami ide hashing sebagai suatu teknik pencarian
- Memahami dan mengimplementasikan tipe data abstrak hashing.

4.1 Pendekatan Rekursif

Rekursi adalah proses pengulangan sesuatu dengan cara kesamaan-diri. Sebagai contohnya, saat dua cermin berada paralel antara satu dengan yang lain, gambar yang tertangkap adalah suatu bentuk rekursi tak-terbatas. Penggunaan paling umum dari rekursi yaitu dalam ilmu komputer, yang mengacu kepada suatu metode mendefinisikan fungsi dimana fungsi tersebut memanggil definisinya sendiri. Bilangan Fibonacci adalah contoh klasik dari rekursi:

- Fib(0) adalah 0 [kasus dasar]
- Fib(1) adalah 1 [kasus dasar]
- Untuk semua integer n > 1: Fib(n) adalah (Fib(n-1) + Fib(n-2)) [definisi rekursif]

Listing 4.1 Fungsi iteratif untuk menghitung hasil penjumlahan bilangan di dalam suatu list

```
def listsum(numList):
   theSum = 0
   for i in numList:
      theSum = theSum + i
   return theSum

print(listsum([1,3,5,7,9]))
```

Listing 4.2 Penjumlahan dalam bentuk rekursi.

```
def listsum(numList):
  if len(numList) == 1:
      return numList[0]
  else:
      return numList[0] + listsum(numList[1:])

print(listsum([1,3,5,7,9]))
```

Bagaimana jika kita akan mengkonversi suatu integer ke string pada suatu basis: biner dan hexadecimal. Misalnya mengubah integer 10 ke tipe stringnya menjadi "10" atau ke bentuk binernya menjadi "1010". Ada banyak algoritma untuk menyelesaikan masalah ini, termasuk stack, namun formulasi rekursif dianggap sangat elegan.

Listing 4.3 Implementasi Python untuk mengkonversi integer ke string pada basis 2 atau 16.

```
def toStr(n,base):
  convertString = "0123456789ABCDEF"
  if n < base:
    return convertString[n]
  else:
    return toStr(n//base,base) + convertString[n%base]

print(toStr(1453,16))
```

Listing 4.4 Implementasi rekursi menggunakan stack

```
import Stack # sebagaimana didefinisikan sebelumnya

r_stack = Stack()

def to_str(n, base):
        convert_string = "0123456789ABCDEF"

        while n > 0:
                if n < base:
                        r_stack.push(convert_string[n])
                else:
                        r_stack.push(convert_string[n % base])

        n = n // base
        res = ""

        while not r_stack.is_empty():
                res = res + str(r_stack.pop())

        return res

print(to_str(1453, 16))
```

4.2 Pencarian Item dalam List

Listing 4.5 Pencarian sequential terhadap item data dalam list mengikuti indexnya

```
        def sequentialSearch(alist, item):
           pos = 0
           found = False
```

```
    while pos < len(alist) and not found:
        if alist[pos] == item:
            found = True
        else:
            pos = pos+1

    return found

testlist = [1, 2, 32, 8, 17, 19, 42, 13, 0]
print(sequentialSearch(testlist, 3))
print(sequentialSearch(testlist, 13))
```

Listing 4.6 Pencarian *sequential* terhadap item dalam List yang telah diurutkan secara *ascending* (*Ordered List*)

```
def orderedSequentialSearch(alist, item):
    pos = 0
    found = False
    stop = False

    while pos < len(alist) and not found and not stop:
        if alist[pos] == item:
            found = True
        else:
            if alist[pos] > item:
                stop = True
            else:
                pos = pos+1

    return found

testlist = [0, 1, 2, 8, 13, 17, 19, 32, 42,]
print(orderedSequentialSearch(testlist, 3))
print(orderedSequentialSearch(testlist, 13))
```

Listing 4.7 Pencarian Biner pada Ordered List

```
def binarySearch(alist, item):

    first = 0
    last = len(alist)-1
    found = False

    while first <= last and not found:
        midpoint = (first + last)//2

        if alist[midpoint] == item:
            found = True
        else:
            if item < alist[midpoint]:
```

```
            last = midpoint-1
        else:
            first = midpoint+1

    return found

testlist = [0, 1, 2, 8, 13, 17, 19, 32, 42,]
print(binarySearch(testlist, 3))
print(binarySearch(testlist, 13))
```

Listing 4.8 Fungsi pencarian biner dalam bentuk rekursif

```
def binarySearch(alist, item):

    if len(alist) == 0:
        return False
    else:
        midpoint = len(alist) // 2
        if alist[midpoint] == item:
          return True
        else:
          if item < alist[midpoint]:
            return binarySearch(alist[ : midpoint], item)
          else:
            return binarySearch(alist[midpoint+1 : ], item)

testlist = [0, 1, 2, 8, 13, 17, 19, 32, 42,]
print(binarySearch(testlist, 3))
print(binarySearch(testlist, 13))
```

4.3 Pengurutan

Listing 4.9 Fungsi pengurutan item-item dalam list menggunakan pendekatan *bubble*.

```
def bubbleSort(alist):

   for passnum in range(len(alist)-1,0,-1):
      for i in range(passnum):
         if alist[i]>alist[i+1]:
            temp = alist[i]
            alist[i] = alist[i+1]
            alist[i+1] = temp

alist = [54,26,93,17,77,31,44,55,20]
bubbleSort(alist)
print(alist)
```

Listing 4.10 Fungsi pengurutan buble yang lebih cepat (Short Bubble Sort)

```
def shortBubbleSort(alist):

   exchanges = True
   passnum = len(alist)-1
   while passnum > 0 and exchanges:
      exchanges = False

      for i in range(passnum):
         if alist[i]>alist[i+1]:
            exchanges = True
            temp = alist[i]
            alist[i] = alist[i+1]
            alist[i+1] = temp
      passnum = passnum-1

alist=[20,30,40,90,50,60,70,80,100,110]
shortBubbleSort(alist)
print(alist)
```

Listing 4.11 Fungsi quicksort untuk mengurutkan item-item di dalam List

```
def quickSort(alist):
  quickSortHelper(alist,0,len(alist)-1)

def quickSortHelper(alist,first,last):

  if first<last:
     splitpoint = partition(alist,first,last)
     quickSortHelper(alist,first,splitpoint-1)
     quickSortHelper(alist,splitpoint+1,last)

def partition(alist,first,last):

  pivotvalue = alist[first]
  leftmark = first+1
  rightmark = last

 done = False
  while not done:

     while leftmark <= rightmark and alist[leftmark] <= pivotvalue:
        leftmark = leftmark + 1

     while alist[rightmark] >= pivotvalue and rightmark >= leftmark:
        rightmark = rightmark -1

     if rightmark < leftmark:
        done = True
```

```
        else:
            temp = alist[leftmark]
            alist[leftmark] = alist[rightmark]
            alist[rightmark] = temp

    temp = alist[first]
    alist[first] = alist[rightmark]
    alist[rightmark] = temp

    return rightmark

alist = [54,26,93,17,77,31,44,55,20]
quickSort(alist)
print(alist)
```

4.4 Rangkuman

Kita sudah melihat contoh dari beberapa algoritma rekursif. Algoritma-algoritma ini dipilih untuk mengekspos beberapa masalah berbeda dimana rekursi adalah teknik penyelesaian masalah yang efektif. Poin kunci untuk diingat dari lab 4 ini adalah

- Semua algoritma rekursif harus mempunyai suatu kasus basis (*base case*).
- Suatu algoritma rekursif harus mengubah statusnya dan membuat progress menuju *base case*.
- Suatu algoritma rekursif harus memanggil dirinya sendiri (secara rekursif).
- Rekursi menggantikan iterasi dalam beberapa kasus.
- Algoritma rekursif sering diformalkan ke ekspresi formal dari masalah yang coba diselesaikan.
- Rekursi bukan selalu jawaban. Kadang solusi rekursi komputasinya lebih mahal daripada algoritma alternatif.

4.5 Tugas Pendahuluan

1. Tuliskan suatu fungsi yang inputnya suatu string dan mengembalikan string baru yang merupakan kebalikan dari string inputan.
2. Buatlah suatu fungsi yang menerima input suatu string dan mengembalikan True jika string tersebut bersifat palindrome, False jika tidak. Berikut ini adalah beberapa contoh teks atau kata yang palindrom:

   madam i'm adam
   radar
   aibohphobia
   Live not on evil
   Reviled did I live, said I, as evil I did deliver
   Go hang a salami; I'm a lasagna hog.
   Able was I ere I saw Elba
   Kanakanak – kota kecil di Alaska
   Wassamassaw – kota kecil di Dakota Selatan

4.6 Tugas Lanjutan

3. Implementasikan fungsi *mergeSort* tanpa mengunakan operator *slice*.
4. Tulislah suatu fungsi rekursif untuk membalikkan suatu list.
5. Tulislah suatu program untuk menyelesaikan masalah berikut: Anda mempunyai dua ember, masing-masing berukuran 4 liter dan 3 liter. Tidak ada tanda atau tulisan apapun di kedua ember tersebut. Ada keran yang dapat digunakan untuk mengisikan air ke dalam ember. Bagaimana caranya agar ember 4 liter berisi air tepat 2 liter?

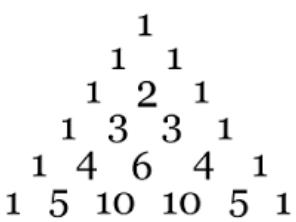
4.7 Latihan Mandiri

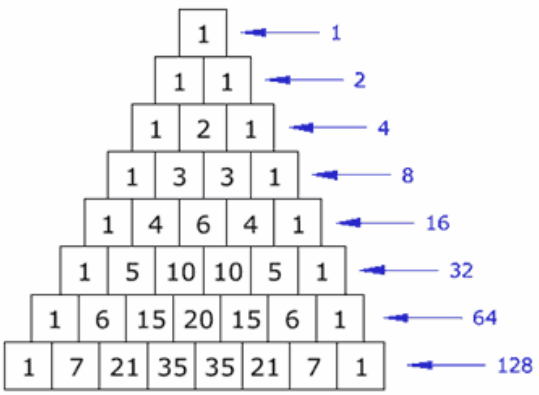
6. Implementasikan suatu solusi untuk masalah Menara Hanoi menggunakan tiga stack untuk memegang piringan.
7. Tulislah suatu fungsi rekursif untuk menyelesaikan faktorial dari suatu bilangan.
8. Tuliskan suatu fungsi rekursif untuk menyelesaikan deret Fibonacci. Bagaimana kinerja dari fungsi rekursif ini dibandingkan dengan versi iteratifnya?
9. Generalkan masalah di atas (ember 4 dan 3 liter) sehingga parameter terhadap solusi memasukkan ukuran dari setiap ember dan jumlah liter air akhir yang tersisa di dalam ember yang lebih besar.
10. Tuliskan suatu program yang memecahkan masalah 3 mahasiswi dan 3 preman yang datang ke suatu tepi sungai dan menemukan satu boat yang hanya mampu menampung dua orang. Setiap orang harus menyeberang sungai dan melanjutkan perjalanan. Namun, jika preman jumlahnya melebihi mahasiswi pada suatu sisi sungai, maka preman akan menyakiti mahasiswi. Temukan serangkaian cara menyeberang sungai yang aman bagi mahasiswi sehingga semuanya (6 orang tersebut) dapat menyeberang sungai.

11. Segitiga Pascal adalah suatu segitiga bilangan dengan angka-angka disusun dalam baris-baris mengikuti rumus:

    $$anr = n!r!\,(n-r)!$$

    Persamaan ini bersifat *binomial coefficient*. Kita dapat membangun segitiga Pascal dengan menambahkan dua bilangan yang secara diagonal di atas bilangan dalam segitiga. Agar jelas, perhatikan dua gambar di bawah ini:





Tuliskan suatu program yang mencetak segitiga Pascal. Program sebaiknya dapat menerima suatu parameter yang memberitahukan berapa banyak baris yang akan dicetak.

12. Siapkan suatu eksperimen acak (*random experiment*) untuk menguji perbedaan antara *sequential search* dan *binary search* pada suatu list integer.
13. Gunakan fungsi pencarian biner yang telah diberikan (pendekatan rekursif dan iteratif) sebelumnya. Bangkitkan suatu ordered list yang berisi integer acak dan lakukan suatu analisa *benchmark* untuk setiap pendekatan. Apa hasilnya, jelaskan!
14. Implementasikan suatu binary search menggunakan rekursi tanpa operator slice. Ingat kembali bahwa anda perlu melewatkan list bersama dengan nilai index starting dan ending untuk sublist tersebut. Bangkitkan suatu ordered list yang berisi integer acak dan lakukan suatu analisa *benchmark*.
15. Implementasikan metode len (__len__) untuk mewujudkan ADT Map tabel hash.
16. Implementasikan metode in (__contains__) untuk ADT Map tabel hash.
17. Bagaimana anda dapat menghapus item-item dari suatu hash table yang menggunakan *chaining* untuk resolusi kolisinya? Bagaimana jika digunakan *open addressing*? Keadaan khusus apa yang harus ditangani? Implementasikan metode del untuk kelas HashTable.
18. Dalam implementasi map *hash table*, ukuran *hash table* dipilih 101. Jika tabel penuh, ini perlu ditingkatkan. Tuliskan ulang dari metode put sehingga tabel akan berubah ukuran (*resize*) secara otomatis ketika *loading factor* mencapai suatu nilai yang telah ditentukan (anda dapat menentukan nilai ini berdasarkan pada taksiran anda tentang beban vs. kinerja).
19. Implementasikan *bubble sort* menggunakan *simultaneous assignment*.
20. Implementasikan *selection sort* menggunakan *simultaneous assignment*.
21. Satu cara meningkatkan quick sort adalah menggunakan insertion sort pada lists yang panjangnya kecil (disebut "*partition limit*"). Tulis ulang kode quick sort dan gunakan pendekatan ini untuk mengurutkan list yang berisi integer random.

Lab 5: Tree dan Algoritmanya

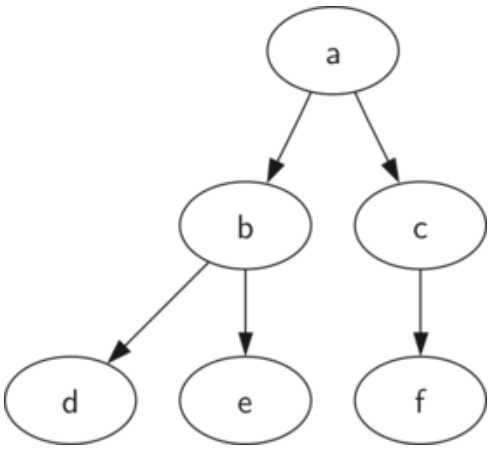
5.0 Obyektif

- Memahami apa itu struktur data tree dan bagaimana pemanfaatannya.
- Mengerti bagaimana tree digunakan untuk mengimplementasikan suatu struktur data map.
- Mengimplementasikan tree menggunakan suatu list.
- Mengimplementasikan tree menggunakan kelas dan referensi.
- Mengimplementasikan tree sebagau suatu struktur data rekursif.
- Mengimplementasikan suatu antrian prioritas (*priority queue*) menggunakan heap.

5.1 Tree Berbasis List

Tree digunakan di banyak bidang informatika termasuk sistem operasi, pemrosessan bahasa alami, penggalian data web, grafika, sistem database dan jaringan komputer. Struktur data tree mempuyai suatu ***root*** (akar), ***branches*** (cabang) dan ***leaves*** (daun-daun).

Pada tree berbentuk *list of lists* (list di dalam list), kita menyimpan nilai dari node root sebagai elemen pertama dari list. Elemen kedua dari list akan merepresentasikan sub-tree kiri. Elemen ketiga adalah list lain yang mewakili sub-tree kanan. Tentang ilustrasikan struktur data ini, gambar 5.1 memperlihatkan tree sederhana beserta list yang mewakilinya.




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

Gambar 5.1 Tree dan List yang Merepresentasikannya (sisi kanan)

Kita dapat mengakses subtree dari list menggunakan indexing list standard. Berikut ini adalah contohnya:

```
myTree = ['a', ['b', ['d',[],[]], ['e',[],[]] ], ['c', ['f',[],[]], []] ]
print(myTree)
print('Subtree kiri =  ', myTree[1])
print('Root =  ',  myTree[0])
print('Subtree kanan =  ', myTree[2])
```

Definisi dari struktur data Tree tersebut dapat diformalkan dengan menyediakan beberapa fungsi yang membuatnya lebih mudah digunakan. Metode BinaryTree membuat suatu list yang menyerupai binary tree kosong:

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

Fungsi BinaryTree di atas membangun suatu list dengan node root dan dua sublist kosong untuk node anak. Untuk menambahkan suatu subtree kiri ke root dari tree, dilakukan penyisipan list baru ke dalam posisi kedua dari list root.

Listing 5.1 Kode Python untuk melakukan penyisipan anak kiri (*left child*)

```
def insertLeft(root, newBranch):
    t = root.pop(1)
    if len(t) > 1:
        root.insert(1, [newBranch, t, [] ])
    else:
        root.insert(1, [newBranch, [], [] ])

    return root
```

Perhatikan, bahwa untuk menyisipkan suatu anak kiri, kita harus mendapatkan list (mungkin kosong) yang berkaitan dengan anak kiri sekarang. Kita kemudian menambahkan anak kiri baru, menginstal anak kiri lama sebagai anak kiri dari yang baru. Ini memungkinkan kita untuk menyambung suatu node baru ke dalam tree pada posisi tertentu. Kode insertRigt() sama dengan insertLeft(), diperlihatkan pada Listing 5.2.

Listing 5.2 Kode Python untuk menambahkan anak kanan baru (*right child*)

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

    return root
```

Listing 5.3 Fungsi untuk mengambil dan mengubah nilai root serta mengambil nilai anak kiri dan kanan

```
def getRootVal(root):
    return root[0]

def setRootVal(root, newVal):
    root[0] = newVal

def getLeftChild(root):
    return root[1]
```

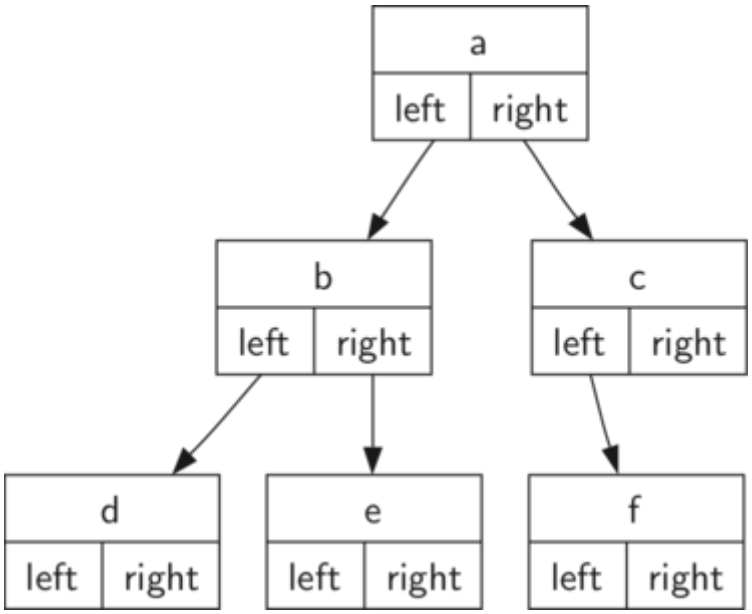
```
def getRightChild(root):
        return root[2]
```

Listing 5.4 Contoh Pembuatan BynaryTree dan pemanfaatannya

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

def insertLeft(root,newBranch):
   t = root.pop(1)
   if len(t) > 1:
      root.insert(1,[newBranch,t,[]])
   else:
      root.insert(1,[newBranch, [], []])

   return root

def insertRight(root,newBranch):
   t = root.pop(2)
   if len(t) > 1:
      root.insert(2, [newBranch,[],t])
   else:
      root.insert(2, [newBranch,[],[]])

   return root

def getRootVal(root):
   return root[0]

def setRootVal(root,newVal):
   root[0] = newVal

def getLeftChild(root):
   return root[1]

def getRightChild(root):
   return root[2]

r = BinaryTree(3)
insertLeft(r,4)
insertLeft(r,5)
insertRight(r,6)
insertRight(r,7)
l = getLeftChild(r)
print(l)
setRootVal(l,9)
print(r)
insertLeft(l,11)
print(r)
print(getRightChild(getRightChild(r)))
```

5.2 Tree dengan Node & Referensi

Metode kedua untuk menyajikan tree adalah menggunakan node dan referensi, yaitu dengan mendefinisikan suatu kelas yang mempunyai atribut-atribut untuk nilai root, subtree kiri dan kanan. Representasi demikian dekat dengan paradigma *object-oriented programming*. Di sini, tree distrukturkan seperti pada gambar 5.2.



Gambar 5.2 Tree direpresentasikan menggunakan pendekatan Node dan Referensi

Listing 5.5 Class dasar dari suatu Pohon Biner (Binary Tree)

```
class BinaryTree:
        def __init__(self, rootObj):
                self.key = rootObj
                self.leftChild = None
                self.rightChild = None
```

Listing 5.6 Cara menambahkankan anak baru di sisi kiri

```
def insertLeft(self, newNode):
        if self.leftChild == None:
                self.leftChild = BinaryTree(newNode)
        else:
                t = BinaryTree(newNode)
                t.leftChild = self.leftChild
                self.leftChild = t
```

Listing 5.7 Cara menambahkan node anak di sisi kanan

```
def insertRight(self, newNode):
        if self.rightChild == None:
                self.rightChild = BinaryTree(newNode)
        else:
                t = BinaryTree(newNode)
                t.rightChild = self.rightChild
                self.rightChild = t
```

Listing 5.8 Metode-metode untuk mengatur dan mengambil nilai root, anak kiri dan kanan

```
def getRightChild(self):
        return self.rightChild

def getLeftChild(self):
        return self.leftChild

def setRootVal(self,obj):
        self.key = obj

def getRootVal(self):
        return self.key
```

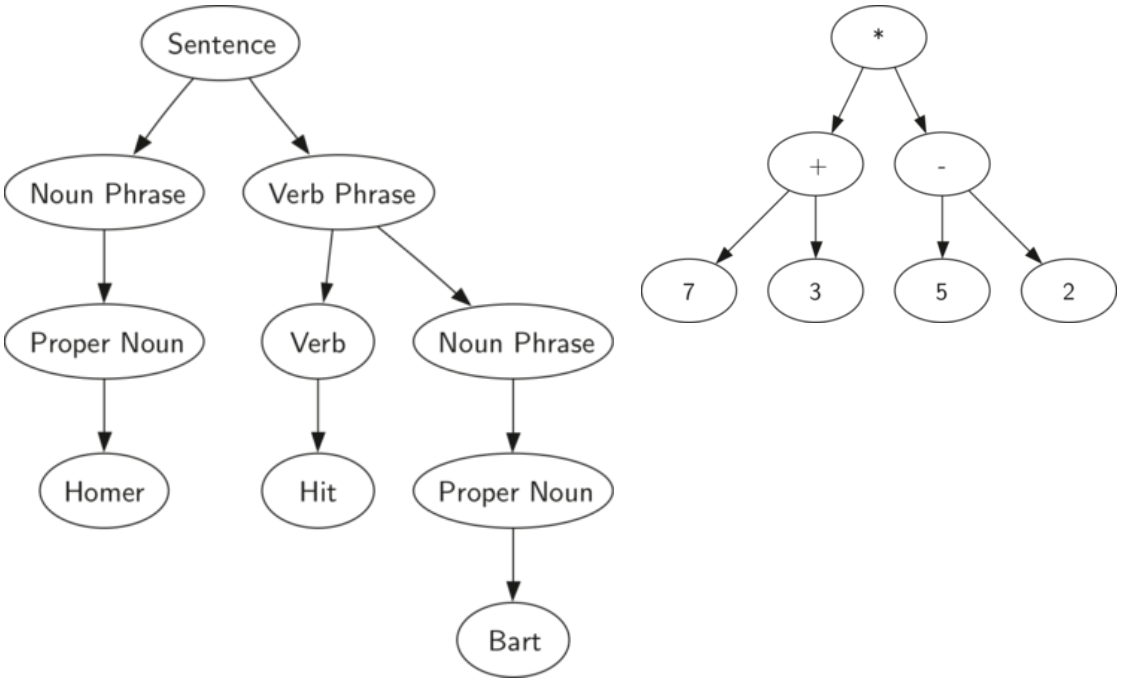
Listing 5.9 Binary Tree dan cara mengaksesnya

```
class BinaryTree:
   def __init__(self,rootObj):
      self.key = rootObj
      self.leftChild = None
      self.rightChild = None

   def insertLeft(self,newNode):
      if self.leftChild == None:
         self.leftChild = BinaryTree(newNode)
      else:
         t = BinaryTree(newNode)
         t.leftChild = self.leftChild
         self.leftChild = t

   def insertRight(self,newNode):
      if self.rightChild == None:
         self.rightChild = BinaryTree(newNode)
      else:
         t = BinaryTree(newNode)
         t.rightChild = self.rightChild
         self.rightChild = t

   def getRightChild(self):
      return self.rightChild
```

```
    def getLeftChild(self):
        return self.leftChild

    def setRootVal(self,obj):
        self.key = obj

    def getRootVal(self):
        return self.key

r = BinaryTree('a')
print(r.getRootVal())
print(r.getLeftChild())
r.insertLeft('b')
print(r.getLeftChild())
print(r.getLeftChild().getRootVal())
r.insertRight('c')
print(r.getRightChild())
```

5.3 Parse Tree

Implementasi tree yang lengkap secara struktur dapat digunakan untuk menyelesaikan malasah ril, di antaranya adalah parse tree (pohon uraian). Gambar 5.3 menjelaskan posisi kalimat “Homer Hit Bart” dan ekspresi matematika (7+3) * (5-2) menggunakan pohon uraian.



Gambar 5.3 Pohon Uraian untuk menganalisa suatu kalimat dan ekspresi matematika

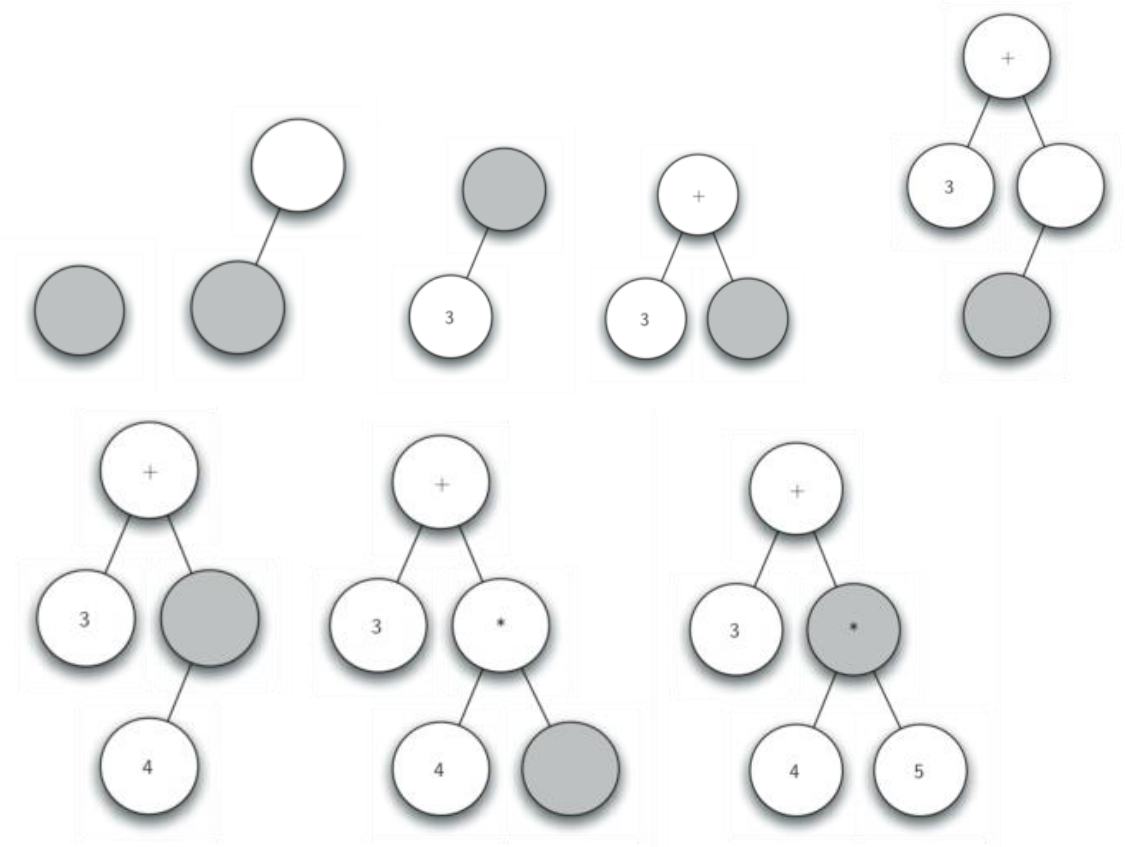
Langkah pertama dalam pembangunan suatu *parse tree* Adalah memecah string ekspresi ke dalam daftar tokens. Ada 4 jenis token berbeda yang perlu diperhatikan: kurung left, kurung kanan, operator dan operan. Kita tahun bahwa kapanpun kita membaca kurung kiri kita

memulai suatu ekspresi baru, dan karena itu kita membuat suatu tree baru untuk menyesuaikan ekspresi tersebut. Sebaliknya, kapanpun kita membaca suatu kurung kanan, kita telah menyelesaikan suatu ekspresi. Kita juda mengetahui bahwa operan akan menjadi node daun (leave) dan anak-anak (children) dari operator-operatornya. Terakhir, kita mengetahui bahwa setiap operator akan mempunyai anak kiri dan kanan.

Berdasarkan informasi di atas kita dapat mendefinisikan 4 rule berikut:

1. Jika *current token* adalah '(', tambahkan node baru sebagai anak kiri (*left child*) dari *current node*, dan turunkan ke anak kiri tersebut.
2. Jika *current token* adalah operator dalam list ['+','-','/','*'], set nilai root dari *current node* ke operator yang direpresentasikan oleh *current token*. Tambahkan node baru sebagai anak nakan dari *current node* dan turunkan ke anak kanan tersebut.
3. Jika *current token* adalah suatu bilangan, set nilai root dari *current node* ke bilangan tersebut dan kembali ke induknya.
4. Jika *current token* adalah ')', pindahkan ke induk dari *current node*.

Sekarang mari kita manfaatkan 4 aturan di atas untuk menguji ekspresi matematika (3 + (4 * 5)). Ekspresi ini akan diurai ke dalam daftar token ['(', '3', '+', '(', '4', '*', '5' ,')',')']. Awalnya dimulai dengan suatu *parse tree* yang mengandung node root kosong. Gambar 5.4 mengilustrasikan struktur dan isi dari *parse tree* saat setiap token baru diproses.



Gambar 5.4 Menyusuri proses pembangunan Pohon Uraian

Mari kita lihat *step-by-step* blusukan yang dilalui oleh ekspresi matematika (3 + (4 * 5)) tersebut:

1. Buat suatu tree kosong.
2. Baca (sebagai token pertama. Berdasarkan rule 1, buat node baru sebagai anak kiri (left child) dari root. Jadikan current node (node aktif) ke anak baru ini.
3. Baca 3 sebagai token berikutnya. Berdasarkan rule 3, set nilai root dari current node ke 3 dan kembali arahkan tree ke induknya.
4. Baca + sebagai token selanjutnya. Berdasarkan rule 2, set nilai root dari current node ke + dan tambahkan node baru sebagai anak kanan (right child). Anak kanan baru ini menjadi current node.
5. Baca (sebagai token berikutnya. Berdasarkan rule 1, buat node baru sebagai anak kiri dari current node. Anak kiri baru ini menjadi current node.
6. Baca 4 sebagai token berikutnya. Berdasarkan rule 3, set nilai dari current node ke 4. Jadikankan induk dari 4 ke current node.
7. Baca * sebagai token selanjutnya. Berdasarkan rule 2, set nilai root dari current node ke * dan buat anak kanan baru. Anak kanan baru ini menjadi current node.
8. Baca 5 sebagai token berikutnya. Berdasarkan rule 3, set nilai root dari current node ke 5. Jadikankan induk dari 5 ke current node.
9. Baca) sebagai token berikutnya. Berdasarkan rule 4 kita buat induk dari * ke current node.
10. Baca) sebagai token berikutnya. Berdasarkan rule 4 jadikan induk dari + ke current node. Pada titik ini tidak ada induk untuk + sehingga dianggap SELESAI.

Listing 5.10: Parse Tree untuk mengevaluasi ekspresi matematika sederhana

```
from pythonds.basic.stack import Stack

from pythonds.trees.binaryTree import BinaryTree

def buildParseTree(fpexp):
    fplist = fpexp.split()
    pStack = Stack()
    eTree = BinaryTree('')
    pStack.push(eTree)
    currentTree = eTree

    for i in fplist:
        if i == '(':
            currentTree.insertLeft('')
            pStack.push(currentTree)
            currentTree = currentTree.getLeftChild()
        elif i not in ['+', '-', '*', '/', ')']:
            currentTree.setRootVal(int(i))
            parent = pStack.pop()
            currentTree = parent
        elif i in ['+', '-', '*', '/']:
            currentTree.setRootVal(i)
            currentTree.insertRight('')
            pStack.push(currentTree)
            currentTree = currentTree.getRightChild()
        elif i == ')':
```

```
        currentTree = pStack.pop()
      else:
        raise ValueError

  return eTree

pt = buildParseTree("( ( 10 + 5 ) * 3 )")
pt.postorder()  #sudah terdefinisi dan dijelaskan kemudian
```

Listing 5.11 Kode untuk mengevaluasi suatu Parse Tree

```
def evaluate(parseTree):
        opers={'+':operator.add,'-':operator.sub,'*':operator.mul,'/':operator.truediv}

        leftC=parseTree.getLeftChild()
        rightC=parseTree.getRightChild()

        if leftC and rightC:
                fn=opers[parseTree.getRootVal()]
                returnfn(evaluate(leftC),evaluate(rightC))
        else:
                returnparseTree.getRootVal()
```

5.4 Pola Penjelajahan Tree

Ada tiga pola umum yang digunkaan untuk mengunjungi semua node dalam tree. Perbedaan antara 3 pola ini adalah urutan kunjungan terhadap node. Kunjungan ini dinamakan "traversal.", yaitu **preorder**, **inorder** dan **postorder**. Berikut ini adalah definisinya:

1. **preorder**
   Dalam suatu preorder traversal, node root dikunjungi pertama, kemudian secara rekursif melakukan preorder traversal dari subtree kiri, diikuti oleh suatu preorder traversal rekursif dari subtree kanan.
2. **inorder**
   Pada suatu inorder traversal, secara rekursif dilakukan inorder traversal pada subtree kiri, mengunjungi node root dan terakhir melakukan inorder traversal rekursif dari subtree kanan.
3. **postorder**
   Dalam suatu postorder traversal, dilakukan suatu postorder traversal rekursif dari subtree kiri dan subtree kanan diikuti dengan kunjungan ke node root.

Listing 5.12 Preorder traversal dari suatu Binay Tree

```
def preorder(tree):
        if tree:
                print(tree.getRootVal())
                preorder(tree.getLeftChild())
                preorder(tree.getRightChild())
```

Listing 5.13 Implementasi preorder sebagai metode dari kelas BinaryTree (metode internal di dalam kelas). Tree diganti dengan self.

```
def preorder(self):
        print(self.key)
        if self.leftChild:
                self.leftChild.preorder()
        if self.rightChild:
                self.rightChild.preorder()
```

Listing 5.14 Algoritma postorder traversal, mirip sekali dengan preorder kecuali pemanggilan print di akhir fungsi

```
def postorder(tree):
        if tree!=None:
                postorder(tree.getLeftChild())
                postorder(tree.getRightChild())
                print(tree.getRootVal())
```

Pemanfaatan umum dari postorder traversal adalah mengevaluasi suatu parse tree. Apa yang dilakukan adalah mengevaluasi subtree kiri, subtree kanan dan menggabungkan keduanya dalam root melalui function call ke suatu operator. Anggapan: binary tree hanya akan menyimpan data tree ekspresi.

Listing 5.15 Evaluasi dengan postOrder

```
def postordereval(tree):
        opers={'+':operator.add,'-':operator.sub,'*':operator.mul,'/':operator.truediv}
        res1=None
        res2=None

        if tree:
                res1=postordereval(tree.getLeftChild())
                res2=postordereval(tree.getRightChild())
        if res1 and res2:
                return opers[tree.getRootVal()](res1,res2)
        else:
                return tree.getRootVal()
```

Listing 5.16 Inorder traversal

```
def inorder(tree):
        if tree!=None:
                inorder(tree.getLeftChild())
                print(tree.getRootVal())
                inorder(tree.getRightChild())
```

Jika kita mengerjakan suatu *inorder traversal* sederhana dari suatu *parse tree* maka diperoleh (kembali) ekspresi originalnya tanpa tanda kurung. Lakukan perubahan terhadap algortima inorder sederhana agar kita mampu menangani versi ekspresi bertanda kurung penuh.

Modifikasinya terhadap template dasar: print suatu kurung kiri sebelum *recursive call* ke subtree kiri dan print kurung kanan setelah *recursive call* terhadap subtree kanan.

Listing 5.17 Perubahan terhadap inorder traversal

```
def printexp(tree):
        sVal=""
        if tree:
                sVal='('+printexp(tree.getLeftChild())
                sVal=sVal+str(tree.getRootVal())
                sVal=sVal+printexp(tree.getRightChild())+')'

        return sVal
```

5.5 Antrian Berprioritas dengan Binary Heaps

Priority Queue adalah variasi dari struktur data FIFO Queue. Dalam priority queue logical order dari item-item di dalam queue ditentukan oleh prioritasnya. Item dengan prioritas tertinggi berada di depan queue dan item berprioritas terendah dibelakang. Sehingga ketika dilakukan enqueue suatu item pada priority queue, item baru may move all the way ke depan.

Priority queue dapat diimplementasikan dengan fungsi pengurutan dan list.Cara yang lebih baik adalah menggunakan binary heap yang memerlukan waktu lebih kecil O(log{n}).

Binary heap mempunyai dua variasi: **min heap**, dimana key paling kecil selalu berada di depan, dan **max heap** yang nilai key terbesarnya selalu diletakkan di depan.

Operasi dasar yang umumnya ada pada suatu Binary Heap adalah:

- binaryheap() untuk membuat suatu binary heap baru yang kosong.
- insert(k) untuk menambahkan suatu item baru ke heap.
- findMin() mengembalikan item dengan nilai key minimum, membiarkan item di dalam heap.
- delMin() mengembalikan item dengan nilai key minimum, menghapus item dari heap.
- isEmpty() mengembalikan true jika heap kosong, false jika tidak.
- size() mengembalikan jumlah item di dalam heap.
- buildHeap(list) membangun suatu heap baru dari suatu daftar key.

Listing 5.18 Implementasi binary heap lengkap

```
class BinHeap:
  def __init__(self):
    self.heapList = [0]
    self.currentSize = 0

  def percUp(self,i):
    while i // 2 > 0:
      if self.heapList[i] < self.heapList[i // 2]:
```

```
            tmp = self.heapList[i // 2]
            self.heapList[i // 2] = self.heapList[i]
            self.heapList[i] = tmp
        i = i // 2

    def insert(self,k):
      self.heapList.append(k)
      self.currentSize = self.currentSize + 1
      self.percUp(self.currentSize)

    def percDown(self,i):
      while (i * 2) <= self.currentSize:
          mc = self.minChild(i)
          if self.heapList[i] > self.heapList[mc]:
              tmp = self.heapList[i]
              self.heapList[i] = self.heapList[mc]
              self.heapList[mc] = tmp
          i = mc

    def minChild(self,i):
      if i * 2 + 1 > self.currentSize:
          return i * 2
      else:
          if self.heapList[i*2] < self.heapList[i*2+1]:
              return i * 2
          else:
              return i * 2 + 1

    def delMin(self):
      retval = self.heapList[1]
      self.heapList[1] = self.heapList[self.currentSize]
      self.currentSize = self.currentSize - 1
      self.heapList.pop()
      self.percDown(1)
      return retval

    def buildHeap(self,alist):
      i = len(alist) // 2
      self.currentSize = len(alist)
      self.heapList = [0] + alist[:]
      while (i > 0):
          self.percDown(i)
          i = i - 1

bh = BinHeap()
bh.buildHeap([9,5,6,2,3])

print(bh.delMin())
print(bh.delMin())
print(bh.delMin())
```

```
print(bh.delMin())
print(bh.delMin())
```

5.6 Binary Search Trees

Kita telah melihat dua cara mendapatkan pasangan key-value dalam suatu koleksi. Koleksi ini mengimplementasikan ADT **map**. Dua implementasi dari ADT map tersebut adalah binary search pada suatu list dan hash tables. **Binary search tree** adalah cara lain untuk memetakan dari suatu key ke nilainya. Berikut ini adalah beberapa operasi yang dapat dilakukan terhadap map:

- map() untuk membuat suatu map baru yang kosong.
- put(key,val) menambahkankan suatu pasangan key-value baru ke map. Jika key telah ada dalam map maka ganti nilai lama dengan nilai yang baru.
- get(key) diberikan suatu key, kembalikan nilai yang disimpan di dalam map atau None jika tidak.
- del menghapus pasangan key-value dari map menggunakan pernyataan berbentuk del map[key].
- len() mengembalikan jumlah pasangan key-value yang disimpan dalam map.
- in mengembalikan True untuk suatu pernyataan dari bentuk key in map, jika key yang diberikan ada dalam map.

Listing 5.19 Contoh Binary Search Tree Lengkap

```
class TreeNode:
  def __init__(self,key,val,left=None,right=None,parent=None):
    self.key = key
    self.payload = val
    self.leftChild = left
    self.rightChild = right
    self.parent = parent

  def hasLeftChild(self):
    return self.leftChild

  def hasRightChild(self):
    return self.rightChild

  def isLeftChild(self):
    return self.parent and self.parent.leftChild == self

  def isRightChild(self):
    return self.parent and self.parent.rightChild == self

  def isRoot(self):
    return not self.parent

  def isLeaf(self):
    return not (self.rightChild or self.leftChild)
```

```
    def hasAnyChildren(self):
        return self.rightChild or self.leftChild

    def hasBothChildren(self):
        return self.rightChild and self.leftChild

    def replaceNodeData(self,key,value,lc,rc):
        self.key = key
        self.payload = value
        self.leftChild = lc
        self.rightChild = rc
        if self.hasLeftChild():
            self.leftChild.parent = self
        if self.hasRightChild():
            self.rightChild.parent = self

class BinarySearchTree:

    def __init__(self):
        self.root = None
        self.size = 0

    def length(self):
        return self.size

    def __len__(self):
        return self.size

    def put(self,key,val):
        if self.root:
            self._put(key,val,self.root)
        else:
            self.root = TreeNode(key,val)
        self.size = self.size + 1

    def _put(self,key,val,currentNode):
        if key < currentNode.key:
            if currentNode.hasLeftChild():
                self._put(key,val,currentNode.leftChild)
            else:
                currentNode.leftChild = TreeNode(key,val,parent=currentNode)
        else:
            if currentNode.hasRightChild():
                self._put(key,val,currentNode.rightChild)
            else:
                currentNode.rightChild = TreeNode(key,val,parent=currentNode)

    def __setitem__(self,k,v):
      self.put(k,v)
```

```
def get(self,key):
  if self.root:
     res = self._get(key,self.root)
     if res:
          return res.payload
     else:
          return None
  else:
     return None

def _get(self,key,currentNode):
  if not currentNode:
     return None
  elif currentNode.key == key:
     return currentNode
  elif key < currentNode.key:
     return self._get(key,currentNode.leftChild)
  else:
     return self._get(key,currentNode.rightChild)

def __getitem__(self,key):
  return self.get(key)

def __contains__(self,key):
  if self._get(key,self.root):
     return True
  else:
     return False

def delete(self,key):
 if self.size > 1:
    nodeToRemove = self._get(key,self.root)
    if nodeToRemove:
       self.remove(nodeToRemove)
       self.size = self.size-1
    else:
       raise KeyError('Error, key not in tree')
 elif self.size == 1 and self.root.key == key:
    self.root = None
    self.size = self.size - 1
 else:
    raise KeyError('Error, key not in tree')

def __delitem__(self,key):
  self.delete(key)

def spliceOut(self):
  if self.isLeaf():
     if self.isLeftChild():
          self.parent.leftChild = None
```

```
      else:
          self.parent.rightChild = None
   elif self.hasAnyChildren():
      if self.hasLeftChild():
          if self.isLeftChild():
            self.parent.leftChild = self.leftChild
          else:
            self.parent.rightChild = self.leftChild
          self.leftChild.parent = self.parent
      else:
          if self.isLeftChild():
            self.parent.leftChild = self.rightChild
          else:
            self.parent.rightChild = self.rightChild
          self.rightChild.parent = self.parent

def findSuccessor(self):
  succ = None
  if self.hasRightChild():
     succ = self.rightChild.findMin()
  else:
     if self.parent:
          if self.isLeftChild():
            succ = self.parent
          else:
            self.parent.rightChild = None
            succ = self.parent.findSuccessor()
            self.parent.rightChild = self
  return succ

def findMin(self):
  current = self
  while current.hasLeftChild():
     current = current.leftChild
  return current

def remove(self,currentNode):
    if currentNode.isLeaf(): #leaf
      if currentNode == currentNode.parent.leftChild:
         currentNode.parent.leftChild = None
      else:
         currentNode.parent.rightChild = None
    elif currentNode.hasBothChildren(): #interior
      succ = currentNode.findSuccessor()
      succ.spliceOut()
      currentNode.key = succ.key
      currentNode.payload = succ.payload

    else: # this node has one child
      if currentNode.hasLeftChild():
```

```
            if currentNode.isLeftChild():
               currentNode.leftChild.parent = currentNode.parent
               currentNode.parent.leftChild = currentNode.leftChild
            elif currentNode.isRightChild():
               currentNode.leftChild.parent = currentNode.parent
               currentNode.parent.rightChild = currentNode.leftChild
            else:
               currentNode.replaceNodeData(currentNode.leftChild.key,
                           currentNode.leftChild.payload,
                           currentNode.leftChild.leftChild,
                           currentNode.leftChild.rightChild)
         else:
            if currentNode.isLeftChild():
               currentNode.rightChild.parent = currentNode.parent
               currentNode.parent.leftChild = currentNode.rightChild
            elif currentNode.isRightChild():
               currentNode.rightChild.parent = currentNode.parent
               currentNode.parent.rightChild = currentNode.rightChild
            else:
               currentNode.replaceNodeData(currentNode.rightChild.key,
                           currentNode.rightChild.payload,
                           currentNode.rightChild.leftChild,
                           currentNode.rightChild.rightChild)

mytree = BinarySearchTree()
mytree[3]="red"
mytree[4]="blue"
mytree[6]="yellow"
mytree[2]="at"

print(mytree[6])
print(mytree[2])
```

5.7 Rangkuman

- Struktur data tree memungkinkan kita menulis banyak algoritma menarik. Kita telah melihat algoritma yang menggunakan tree untuk melakukan hal berikut:
- Suatu pohon biner (binary tree) untuk mem-parsing dan mengevaluasi ekspresi.
- Binary tree untuk mengimplementasikan ADT map.
- Pohon biner berimbang (AVL tree) untuk implementasi ADT map.
- Binary tree untuk mengimplementasikan suatu min heap.
- Min heap digunakan untuk implementasi suatu priority queue.

5.8 Tugas Pendahuluan

1. Perbaiki fungsi `buildParseTree` agar mampu menangani ekspresi matematis yang tidak punya spasi antara setiap karakter.

5.9 Tugas Lanjutan

2. Lakukan perubahan terhadap fungsi buildParseTree dan evaluate agar mampu menangani pernyataan boolean (and, or dan not). Ingat bahwa “not” adalah operator uner (*unary*) sehingga kode program sedikit lebih rumit.
3. Lakukan perubahan pada implementasi binary search tree di atas sehingga mampu menangani duplikasi kunci (*key*) dengan tepat. Jika suatu key sudah ada di dalam tree maka *payload* baru akan menggantikan yang lama, bukan menambahkan node lain dengan key yang sama.

5.10 Latihan Mandiri

4. Menggunakan metode findSuccessor, tulislah suatu *non-recursive inorder traversal* untuk suatu *binary search tree*.
5. Lakukan perubahan terhadap kode untuk *binary search tree* agar ber-thread. Tulislah suatu metode *non-recursive inorder traversal* untuk *threaded binary search tree*. Suatu *threaded binary tree* memelihara referensi dari setiap node ke suksesornya.
6. Buat suatu binary heap dengan ukuran heap terbatas. Dengan kata lain, heap hanya memegang track dari n item paling penting. Jika heap meningkatkan ukurannya lebih dari n item maka item yang kurang penting dihilangkan.
7. Bersihkan fungsi printexp sehingga tidak menyertakan “tambahan” himpunan kurung di sekitar setiap bilangan.
8. Menggunakan metode buildHeap, tuliskan suatu fungsi pengurutan yang dapat mengurutkan suatu list dalam waktu O(nlog{n}).
9. Implementasikan suatu binary heap sebagai max heap.
10. Menggunakan kelas BinaryHeap, implementasikan suatu kelas baru bernama PriorityQueue. Kelas PriorityQueue ini akan mengimplementasikan konstruktor, ditambah metode enqueue dan dequeue.