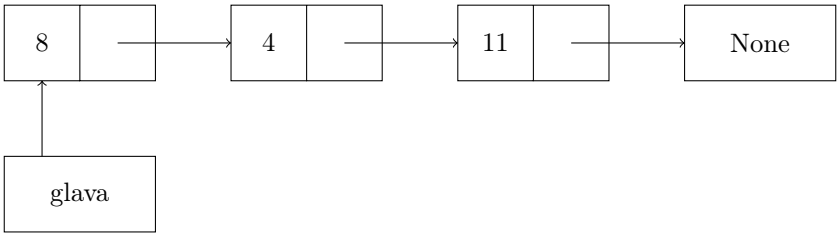
# METODIKA NASTAVE RAČUNARSTVA A
# Strukture podataka – jednostruko povezana lista, binarno pretraživačko drvo, stek, red i graf

Stefan Mišković

**1.** Implementirati klasu `PovezanaLista` koja podržava osnovne metode za rad sa jednostruko povezanim listama: ispis liste, dodavanje na kraj liste, pretraga elemenata i brisanje elemenata iz liste.

*Rešenje.* Pomoćna klasa povezane liste je `Cvor`, koja predstavlja jedan njen element. Svaki element povezane liste je čvor koji se sastoji od dva polja – jedno polje predstavlja njegovu vrednost, a drugo pokazivač na naredni element liste. Pri inicijalizaciji, trenutni čvor pokazuje na `None`. Lista sa slike se sastoji od tri čvora, koji redom imaju vrednosti 8, 4 i 11. Glava liste je čvor čija je vrednost 8, a poslednji čvor, čija je vrednost 11, pokazuje na `None`. Šematski prikaz liste je `glava` $\rightarrow 8 \rightarrow 4 \rightarrow 11 \rightarrow$ `None`.




Promenljiva klase `PovezanaLista` je `glava`, koja uvek treba da pokazuje na početak liste. Pri inicijalizaciji prazne liste, glava pokazuje na `None`. Vrednost glave se eventualno može promeniti dodavanjem novih ili brisanjem postojećih elemenata iz liste. Implementirani su sledeći klasni metodi:

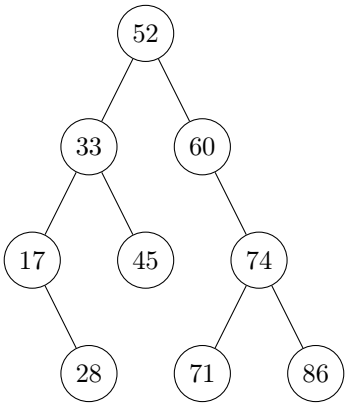
- Metod `ispisi` ispisuje vrednosti elemenata liste, počev od prvog, na kog pokazuje glava. Vrednost trenutnog elementa se u svakoj iteraciji menja na osnovu pokazivača na naredni element. Ovaj proces se ponavlja sve dok trenutni element ne postane `None`.
- Metod `dodaj` dodaje element na kraj liste. Ukoliko je lista prazna, kreira se novi čvor i glava se postavlja na novododatog elementa. Inače, u ciklusu se kreće kroz listu sve dok naredni element nije `None`, odnosno dok se ne stigne do poslednjeg elementa liste. Tada se kreira novi čvor, novododati čvor pokazuje na `None`, a prethodni čvor koji je bio poslednji pokazuje na novododati.
- U metodu `pronadji` se prolazi kroz elemente liste na sličan način kao u metodu `ispisi`. Ukoliko trenutni element liste odgovara zadatom broju, povratna vrednost je `True`. Inače, ako se stiglo do kraja liste, povratna vrednost je `False`.
- Metod `obrisi` briše element iz liste sa zadatom vrednošću, ukoliko postoji. Postrebno je pramtiti vrednosti trenutnog i prethodnog elementa. Prvi se inicijalizuje tako da bude glava liste, a drugi tako da pokazuje na `None`. Ukoliko je vrednost trenutnog čvora jednaka zadatom broju, razlikuju se dva slučaja. Ako se briše glava liste, potrebno je izvršiti ažuriranje tako da glava pokazuje na naredni element od trenutnog. Inače, trenutni element se briše iz liste menjanjem pokazivača prethodnog elementa, tako da pokazuje na čvor na koji je pokazivao trenutni element.

```
class Cvor:
    def __init__(self, vrednost):
        self.vrednost = vrednost
        self.sledeci = None

class PovezanaLista:
    def __init__(self):
        self.glava = None

    def ispisi(self):
        trenutni = self.glava
        while trenutni != None:
            print(trenutni.vrednost, end = "␣")
            trenutni = trenutni.sledeci
        print()

    def dodaj(self, broj):
        trenutni = self.glava
        if trenutni != None:
            while trenutni.sledeci != None:
                trenutni = trenutni.sledeci
            trenutni.sledeci = Cvor(broj)
        else:
            self.glava = Cvor(broj)

    def pronadji(self, broj):
        trenutni = self.glava
        while trenutni != None:
            if trenutni.vrednost == broj:
                return True
            trenutni = trenutni.sledeci
        return False

    def obrisi(self, broj):
        prethodni = None
        trenutni = self.glava
        while trenutni != None:
            if trenutni.vrednost == broj:
                if prethodni == None:
                    self.glava = trenutni.sledeci
                else:
                    prethodni.sledeci = trenutni.sledeci
                return True
            else:
                prethodni = trenutni
                trenutni = trenutni.sledeci
        return False


lista = PovezanaLista()
for i in range(5, 15):
    lista.dodaj(i)
lista.ispisi()
for i in range(20):
```

```
    if lista.pronadji(i):
        print("Broj %d se nalazi u povezanoj listi." % i)
    else:
        print("Broj %d se ne nalazi u povezanoj listi." % i)
for i in range(20):
    if i % 2 == 0:
        continue;
    lista.obrisi(i)
lista.ispisi()
```

**2.** Implementirati klasu `BinarnoDrvo` koja podržava osnovne metode za rad sa binarnim pretraživačkim drvetom: ifinksni ispis drveta, dodavanje novog elementa i proveru da li se čvor sa zadatom vrednošću nalazi u drvetu. Pretpostaviti da su sve vrednosti čvorova u drvetu različite.

*Rešenje.* Na slici je prikazano binarno pretraživačko drvo čiji je koren sa vrednošću 52. Listovi ovog drveta imaju vrednosti 28, 45, 71 i 86. Čvorovi sa vrednostima 52, 33 i 74 imaju po dva potomka, čvorovi sa vrednostima 17 i 60 po jednog potomka, a listovi nemaju potomke. Svaki čvor ima pokazivač na levo i desno poddrvo. Vrednosti svih čvorova levog poddrveta su manji od vrednosti razmatranog čvora, a vrednosti svih čvorova desnog poddrveta su veći od vrednosti razmatranog čvora.




Nad binarnim pretraživačkim drvetom su implementirane funkcije `dodaj`, `ispisi` i `pronadji`, koje pozivaju pomoćne rekurzivne funkcije `dodajPomocna`, `ispisiPomocna` i `pronadjiPomocna`.

- Funkcijom `dodaj` se dodaje novi čvor u drvo. Ukoliko je vrednost novog čvora manja od vrednosti trenutno razmatranog, posmatra se levo poddrvo, a ukoliko je veća, posmatra se desno poddrvo. Kada se naiđe na čvor čiji odgovarajući potomak ne postoji, novododati čvor se smešta na njegovo mesto, tako što se promeni vrednost pokazivača tog čvora sa `None` na novododati.
- Funkcijom `ispisi` se infiksno ispisuju vrednosti drveta. Najpre se vrši rekurzivan poziv za levo poddrvo, zatim se ispisuje vrednost trenutnog čvora, a zatim se rekurzivno razmatra desno poddrvo. Pozivom ove funkcije za koren stabla, vrednosti čvorova se ispisuju u rastućem poretku. Kod prefiksnog ispisa drveta, najpre se ispisuje vrednost trenutnog čvora, a zatim vrše rekurzivni pozivi za levo i desno poddrvo. Kod postfiksnog ispisa, ispis vrednosti trenutnog čvora se vrši nakon rekurzivnih poziva.
- Funkcijom `pronadji` se proverava da li se zadata vrednost nalazi u drvetu. Ukoliko se naiđe na čvor sa istom vrednošću, čvor je pronađen. Inače, ukoliko se naiđe na čvor `None`, čvor sa zadatom vrednošću nije pronađen. Ukoliko je tražena vrednost manja od vrednosti razmatranog čvora, rekurzivno se razmatra levo poddrvo, a ukoliko je veća, rekurzivno se razmatra desno poddrvo.

```
class Cvor:
    def __init__(self, vrednost):
```

```
        self.vrednost = vrednost
        self.levi = None
        self.desni = None

class BinarnoDrvo:
    def __init__(self):
        self.koren = None

    def dodajPomocna(self, cvor, vrednost):
        if vrednost < cvor.vrednost:
            if cvor.levi:
                self.dodajPomocna(cvor.levi, vrednost)
            else:
                cvor.levi = Cvor(vrednost)
        else:
            if cvor.desni:
                self.dodajPomocna(cvor.desni, vrednost)
            else:
                cvor.desni = Cvor(vrednost)

    def dodaj(self, vrednost):
        if self.koren:
            self.dodajPomocna(self.koren, vrednost)
        else:
            self.koren = Cvor(vrednost)

    def ispisiPomocna(self, cvor):
        if cvor:
            self.ispisiPomocna(cvor.levi)
            print(cvor.vrednost, end = "␣")
            self.ispisiPomocna(cvor.desni)

    def ispisi(self):
        if self.koren:
            self.ispisiPomocna(self.koren)
            print()

    def pronadjiPomocna(self, cvor, broj):
        if cvor:
            if broj == cvor.vrednost:
                return True
            elif broj < cvor.vrednost:
                return self.pronadjiPomocna(cvor.levi, broj)
            else:
                return self.pronadjiPomocna(cvor.desni, broj)
        else:
            return False

    def pronadji(self, broj):
        if self.koren:
            return self.pronadjiPomocna(self.koren, broj)
        else:
            return False
```

```
drvo = BinarnoDrvo()
drvo.dodaj(4)
drvo.dodaj(6)
drvo.dodaj(2)
drvo.dodaj(8)
drvo.dodaj(0)
drvo.ispisi()
for i in range(10):
    if drvo.pronadji(i):
        print("Broj %d se nalazi u binarnom drvetu" % i)
    else:
        print("Broj %d se ne nalazi u binarnom drvetu" % i)
```

**3.** Implementirati klasu `Stek` koja podržava osnovne metode za rad sa stekom: dodavanje elementa na vrh steka, brisanje sa vrha steka i proveru da li je stek prazan.

*Rešenje.* Stek je struktura podataka, koja podržava dve operacije – dodavanje na vrh steka i brisanje sa njegovog vrha. U osnovi steka je niz, pa se dodavanje i brisanje vrši dodavanjem i brisanjem poslednjeg elementa niza. U Pythonu se u tu svrhu koriste funkcije `append` i `pop`. Funkcija `pop` vraća i vrednost obrisanog elementa. Funkcijom `prazan` se na osnovu dužine niza proverava da li je stek prazan.

```
class Stek:
    def __init__(self):
        self.niz = []

    def dodaj(self, vrednost):
        self.niz.append(vrednost)

    def obrisi(self):
        return self.niz.pop()

    def prazan(self):
        if len(self.niz) == 0:
            return True
        else:
            return False

stek = Stek()
print(stek.prazan())
stek.dodaj(10)
stek.dodaj(5)
stek.dodaj(7)
print(stek.niz)
stek.obrisi()
print(stek.niz)
stek.obrisi()
print(stek.niz)
print(stek.prazan())
stek.obrisi()
print(stek.niz)
print(stek.prazan())
```

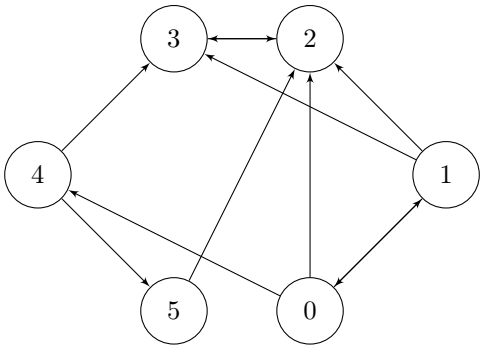
**4.** Implementirati klasu `Red` koja podržava osnovne metode za rad sa redom: dodavanje elementa na kraj reda, brisanje sa početka reda i proveru da li je red prazan.

*Rešenje.* Red je struktura podataka, koja podržava dve operacije – dodavanje na kraj reda i brisanje sa njegovog početka. U osnovi steka je niz, pa se dodavanje i brisanje vrši dodavanjem na kraj niza i brisanjem njegovog prvog elementa. U Pythonu se u tu svrhu koriste funkcije `append` i `pop`. Pozivom `pop(0)` se vraća i vrednost obrisanog prvog elementa. Drugi način brisanja prvog elementa niza je razmatranje intervala `niz[1:]`. Funkcijom `prazan` se na osnovu dužine niza proverava da li je red prazan.

```
class Red:
    def __init__(self):
        self.niz = []

    def dodaj(self, vrednost):
        self.niz.append(vrednost)

    def obrisi(self):
        # return self.niz.pop(0)
        vrednost = self.niz[0]
        if len(self.niz) == 1:
            self.niz = []
        else:
            self.niz = self.niz[1:]
        return vrednost

    def prazan(self):
        if len(self.niz) == 0:
            return True
        else:
            return False

red = Red()
print(red.prazan())
red.dodaj(10)
red.dodaj(5)
red.dodaj(7)
print(red.niz)
red.obrisi()
print(red.niz)
red.obrisi()
print(red.niz)
print(red.prazan())
red.obrisi()
print(red.niz)
print(red.prazan())
```

**5.** Implementirati klasu `Graf` koja podržava osnovne metode pretrage netežinskog usmerenog grafa – pretragu po širini i pretragu po dubini.

*Rešenje.* Graf $G = (V, E)$ je struktura podataka sa skupom čvorova $V$ i ivicama između njih $E$. Graf može biti usmeren, gde ivica ima početni i krajnji čvor, i neusmeren, gde ivica spaja dva čvora u oba smera. Graf može biti i težinski, gde su svim ivicama dodeljenje određene vrednosti težina. Primer jednog usmerenog grafa koji nije težinski je prikazan na slici:




Pretraga grafa po širini (breadth first search – BFS) vrši pretragu grafa tako što razmatra sve neposećene susede od trenutnog čvora i tim redom ih dodaje u red. U svakoj iteraciji se trenutni čvor briše sa početka reda, a njegovi posećeni susedi dodaju na njegov kraj. Time će najpre biti posećen početni čvor, zatim njegovi susedi, zatim neposećeni susedi njegovih suseda, i tako redom. Pretraga grafa po dubini (depth first search – DFS) vrši pretragu tako što rekurzivno razmatra neposećene susede trenutno razmatranog čvora. Time se, u duhu rekurzije, drugi sused razmatranog čvora procesuira tek nakon završenog rekurzivnog poziva za prvog suseda. Graf je u ovom primeru u Pythonu predstavljen nizom nizova.

```
class Graf:
    def __init__(self, matrica):
        self.graf = matrica

    def BFS(self, pocetni):
        posecen = []
        for i in self.graf:
            posecen.append(False)
        red = Red()
        red.dodaj(pocetni)
        posecen[pocetni] = True
        print("Pretraga po sirini: ", end = "")
        while not red.prazan():
            trenutni = red.obrisi()
            print(trenutni, end = " ")
            for i in self.graf[trenutni]:
                if not posecen[i]:
                    red.dodaj(i)
                    posecen[i] = True
        print()

    def DFSpomocna(self, trenutni, posecen):
        posecen[trenutni] = True
        print(trenutni, end = " ")
        for i in self.graf[trenutni]:
            if not posecen[i]:
                self.DFSpomocna(i, posecen)

    def DFS(self, pocetni):
        posecen = []
        for i in self.graf:
            posecen.append(False)
        print("Pretraga po duzini: ", end = "")
        self.DFSpomocna(pocetni, posecen)
        print()
```

```
matrica = [[1, 2, 4], [0, 2, 3], [3], [2], [3, 5], [2]]
graf = Graf(matrica)
graf.BFS(0)
graf.DFS(0)
```