Universität Potsdam
Institut für Informatik

Repetitorium

Aufgaben für die Wiederholung

1 Funktionen und Stackframes

Zeichnen Sie jeweils die Folge von Aufrufstacks mit den Stackframes der folgenden Codes.

Hinweis: Die erste Teilaufgabe entspricht in ihrer Komplexität einer Klausuraufgabe. Die zweite Teilaufgabe stellt ein komplexeres sowie weiterführendes Beispiel dar.

i) Erste Teilaufgabe:

```
1  def f(x,y,z):
2      x += y
3      y = g(x,z) + 1
4      return y
5
6  def g(x,y):
7      x = pow(x,y)
8      return int(pow(x,1/y))
9
10 x = 3
11 y = 2
12 z = 3
13 z = f(x,y,z)
14 print(z)
```

ii) Zweite Teilaufgabe:

```
1  def mtoL(adjM):
2      adjL=[]
3      for zeile in range(len(adjM)):
4          adjL.append([])
5          for spalte in range(len(adjM)):
6              if(adjM[zeile][spalte]==1):
7                  adjL[zeile].append(spalte)
8      return eigg(adjL)
9
10 def eigg(adjL):
11     listeEigg = []
12     for knoten in range(len(adjL)):
13         eiggKnoten = 0
14         for liste in adjL:
15             if knoten in liste:
16                 eiggKnoten+=1
17         listeEigg.append(eiggKnoten)
18     return listeEigg
19
20 adjM=[[1,1],[0,1]]
21 eiggL = mtoL(adjM)
```

2 Funktionale Programmierung

1.1 Beziehen Sie sich bei ihren nachfolgenden Erläuterungen auf die entsprechenden Codezeilen. **Erläutern Sie**, ob das nachfolgende Codesegment die Kriterien der funktionalen Programmierung erfüllt.

```
1  L=[1,2,3,4,5,6,7,8,9,10]
2  for i in range(len(L)):
3      L[i]=L[i]**2
4  for i in L:
5      if i%2!=0:
6          L.remove(i)
7  summe = 5
8  for i in L:
9      summe += i
10 print(summe)
```

1.2 Nutzen Sie bei dieser Aufgabe die drei Funktionen *filter*, *map* und *reduce*. **Passen Sie** den Code aus 1.1 **an**, sodass er die Kriterien der funktionalen Programmierung erfüllt.

2. **Implementieren Sie** den nachfolgenden Code rekursiv.
Hinweis: Die Datenstruktur liste müssen Sie nicht bei der rekursiven Implementierung berücksichtigen.

```
1  def first(xs):
2      return xs[0]
3
4  def rest(xs):
5      return xs[1]
6
7  def map(xs, func):
8      if(xs==()):
9          return xs
10     liste = []
11     while rest(xs)!=():
12         liste.append(func(first(xs)))
13         xs=rest(xs)
14     liste.append(func(first(xs)))
15     xsNew=()
16     for i in liste[::-1]:
17         xsNew =(i,xsNew)
18     return xsNew
19
20 xs=(3,(5,(8,())))
21 xs=map(xs, lambda x: x**2)
22 print(xs)
```

Station 3 – Imperative Programmierung

Entwerfen Sie Algorithmen für die folgenden zwei Probleme. Die geforderten Algorithmen sollen in Pseudocode aufgeschrieben werden. Dieser muss dementsprechend verfeinert werden, dass eine Übertragung in eine imperative Programmiersprache ohne weitere kreative Schritte möglich ist.

1. **Name**: UnidirektionaleKantenZählen
   Eingabe: Gerichteter Graph $G = (V, E)$ in Adjazenzlisten-Repräsentation
   Ausgabe: Anzahl unidirektionaler Kanten[1] in G
2. **Name**: ListenVergleich
   Eingabe: Zwei Listen L_1 und L_2, deren Elemente positive ganze Zahlen sind
   Ausgabe: 1, falls L_2 ein Element enthält, das in L_1 nicht enthalten ist und 0 sonst

[1] Eine Kante $(e_1, e_2) \in E$ heißt unidirektional, wenn $(e_2, e_1) \notin E$.

Station 4 - Graphen

Gegeben sei ein ungerichteter Graph $G = (V, E)$ mit $V = \{a, b, c, d, e, f\}$ durch folgende Adjazenzliste:

```
adj[a] = [b,c,e]        adj[d] = [b,e,f]
adj[b] = [a,c,d]        adj[e] = [a,d]
adj[c] = [a,b]          adj[f] = [d]
```

1. **Stellen** Sie G grafisch **dar**.
2. **Beurteilen Sie**, ob G zusammenhängend ist.
3. **Geben Sie** einen Kreis **an**, der sowohl a als auch d enthält.
4. **Bestimmen Sie** den Abstand zwischen den Knoten c und f.
5. **Geben Sie** die Reihenfolge **an**, in welcher die Knoten durch den Algorithmus „Breitensuche“ aufgesucht werden, wenn die Suche bei b beginnt?
6. Welche Reihenfolge ergibt sich, wenn man stattdessen Tiefensuche verwendet?

Station 5 – Ressourcen

1. **Bestimmen Sie** die Zeitkomplexität im schlechtesten Fall (worst case) für den Algorithmus `lzwPart` bezüglich n = `len(inputString)` als Größe der Eingabe.

```
def lzwPart(inputString):
    lzwList = ["r", "g", "b"]
    pattern, output = "", ""

    for char in inputString:
        if pattern+char in lzwList:
            pattern += char
        else:
            lzwList.append(pattern+char)
            for i in range(0, len(lzwList)):
                if lzwList[i] == pattern:
                    output += str(i)
            pattern = char

    return output+pattern
```

2. Für Schnelle: Betrachten Sie die Funktion $f: \mathbb{N} \to \mathbb{Z}, f(n) = 44n^5 - 40n^4 - 2n^2 - 30$. **Zeigen Sie**, dass $f \in \Omega(n^5)$.