pembahasan trial uas daspro 2023

disusun oleh:

the one and only @kafeyangasli

1. Notasi fungsional untuk sebuah fungsi yang menerima masukan sebuah List L dengan elemen bertipe integer positif, lalu menghasilkan sebuah List baru dengan memfilter elemen dari L yang merupakan bilangan genap

PEMBAHASAN:
Untuk memfilter elemen dari List of integer yang merupakan bilangan genap, kita akan merekursif setiap elemen dalam List, dengan basis ketika List adalah list kosong.

FILTER BILANGAN GENAP **FilterGenap(L)**

DEFINISI DAN SPESIFIKASI

FilterGenap: List of integer ≥ 0 → List of integer ≥ 0

{FilterGenap(L) mengembalikan elemen dari list L yang merupakan bilangan genap.
Basis: L merupakan List kosong
Rekurens 1: Jika FirstElmt(L) habis dibagi 2, Konso FirstElmt dengan rekursif tail.
Rekurens 2: Jika tidak, rekursif tail saja (Elemen tidak masuk ke list baru).

Jika menggunakan FirstElmt, maka rekursifnya menggunakan Tail.
Jika menggunakan LastElmt, maka rekursifnya menggunakan Head.}

REALISASI

```
FilterGenap(L):
   if IsEmpty(L) then
      L {Bisa juga ditulis [] (sama saja, karena L ==[])}
   else
      depend on FirstElmt(L)
         FirstElmt(L) mod 2 = 0 : Konso(FirstElmt(L),FilterGenap(Tail(L)))
         else                   : FilterGenap(Tail(L))
```

REALISASI TANPA DEPEND ON

```
FilterGenap(L):
   if IsEmpty(L) then
      []
   else
      if FirstElmt(L) mod 2 = 0 then
         Konso(FirstElmt(L), FilterGenap(Tail(L)))
      else
         FilterGenap(Tail(L))
```

REALISASI DENGAN KONSI

```
FilterGenap(L):
   if IsEmpty(L) then
      []
   else
      if LastElmt(L) mod 2 = 0 then
         Konsi(FilterGenap(Head(L), LastElmt(L)))
      else
         FilterGenap(Head(L))
```

2. Modifikasi fungsi IsMemberS menjadi IsContainList(S)

PEMBAHASAN:
Dalam soal, kita diminta untuk memodifikasi IsMemberS menjadi IsContainList, dimana fungsi akan bernilai True apabila dalam LoL terdapat setidaknya satu elemen yang merupakan List. Sehingga, kita cukup mengganti hasil dari ekspresi kondisional dalam fungsi IsMemberS pada `depend on` yang kedua menjadi:

`true` jika FirstList(S) adalah sebuah List

`false` jika FirstList(S) adalah sebuah Atom

Lalu, kita `or`-kan dengan rekursif TailList(S), sehingga ketika salah satu elemen bernilai `true`, fungsi akan mengembalikan `true` juga.

APAKAH BERISI LIST **IsContainList(S)**

REALISASI

```
{karena dimintanya modifikasi, asumsikan penggunaan kondisional harus sama persis}
IsContainList(S):
   depend on S
      IsEmpty(S): false
      not IsEmpty(S):
         depend on FirstList(S)
            IsAtom(FirstList(S)): false
            IsList(FirstList(S)): true
               or IsContainList(TailList(S))
```

REALISASI DALAM PYTHON

```
def IsContainList(S):
   if IsEmpty(S):
      return False
   elif not IsEmpty(S):
      if IsAtom(FirstList(S)):
         return False or IsContainList(TailList(S))
      elif IsList(FirstList(S)):
         return True or IsContainList(TailList(S))
```

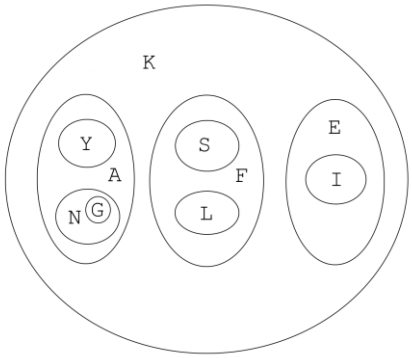
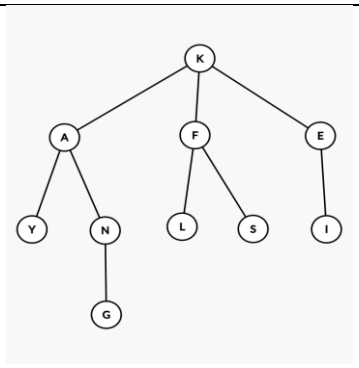
REALISASI DALAM PYTHON TANPA MEMPERHATIKAN REALISASI NOTFUNG

```
def IsContainList(S):
   if IsEmpty(S):
      return False
   else:
      if IsAtom(FirstList(S)):
         return IsContainList(TailList(S))
      elif IsList(FirstList(S)):
         return True
```

3. Pohon N-aire:
    a. Himpunan, Graf, Indentasi
    b. Bentuk Linier
    c. Root dari Pohon
    d. Salah satu successor dari salah satu simpul

PEMBAHASAN:
Jika kita menggunakan nama: **KAFEYANGASLI**
Maka, huruf unik yang ada adalah:

K, A, F, E, Y, N, G, S, L, I

Himpunan	Graf	Indentasi
		K A Y N G F L S E I

Prefix :

(K(A(Y(), N(G())), F(L(), S()), E(I())))

atau

(K (A(Y, N(G)), F(L, S), E(I)))

Infix : (hanya asumsi penulis, sehingga diperlukan koreksi)

(((Y)A(N(G)), (L)F(S)K(E(I)))

Postfix :

((((()Y, (()G)N)A ,(()L, ()S)F, (()I)E)K)

atau

(((Y, (G)N)A, (L, S)F, (I)E)K)

Root Pohon : K

Successor-1 Simpul F: L, S (diperlukan koreksi)

(penyusunan diatas tidak harus sama persis! yang terpenting setidaknya salah satu simpul **harus** memiliki lebih dari dua child)

4. Notasi fungsional Pohon N-aire yang mencari suatu subpohon, lalu mengembalikan List yang berisikan semua simpul, termasuk akar subpohon, dari subpohon tersebut

ELEMEN-ELEMEN DARI SUBPOHON **SubTreeElements(T, X)**

DEFINISI DAN SPESIFIKASI

SubTreeElements: PohonN-aire tidak kosong, elemen → List
{SubTreeElements(T, X) mengembalikan sebuah List yang berisikan simpul-simpul dari suatu subpohon yang dicari.}

RTC: List of PohonN-aire, elemen → List
{RTC(LPN, X) merekursif semua elemen dari LPN untuk dioperasikan terhadap SubTreeElements.}

RTC2: List of PohonN-aire → List
{RTC2(LPN) merekursif semua elemen dari LPN untuk dioperasikan terhadap RENAT.}

RENAT: PohonN-aire tidak kosong → List
{RENAT(T) mengembalikan simpul-simpul dari suatu pohon (Return Nodes of A Tree)}

REALISASI

{asumsikan diperbolehkan menggunakan defspek List sebelumnya di nomor 1, dan terdapat fungsi Konkatenasi}

```
RTC(LPN, X):
   if IsEmpty(LPN) then
      []
   else
      Konkat(SubTreeElements(FirstElmt(LPN), X), RTC(Tail(LPN), X))

RTC2(LPN):
   if IsEmpty(LPN) then
      []
   else
      Konkat(RENAT(FirstElmt(LPN)), RTC2(Tail(LPN)))

RENAT(T):
   if IsTreeEmpty(T) then
      []
   else
      Konkat(Konso(Akar(T), []), RTC2(Anak(T)))

SubTreeElements(T, X):
   depend on T
      IsTreeEmpty(T): []
      not IsTreeEmpty(T):
         if Akar(T) = X then
            RENAT(T)
         else
            RTC(Anak(T), X)
```

APLIKASI

{asumsikan T adalah Tree yang kita buat sebelumnya}

```
→ SubTreeElements(T, "A")
OUTPUT: ["A", "Y", "N", "G"]

→ SubTreeElements(T, "J")
OUTPUT: []
```

mohon maaf saya tidak bisa menjelaskan dengan rinci cara kerja pengkodingannya, hanya bisa memberikan clue bahwa cara kerjanya mirip dengan nbelmt, terutama RTC dengan NbElmtChild.

5. Notasi fungsional BinarySearchTree berisikan integer, lalu mengecek apakah nilai tertinggi dari BST bisa dibagi dengan 4 menggunakan fungsi Lambda untuk mengecek modulo dari nilai tertinggi BST tersebut.

PEMBAHASAN:

Karena ini adalah BST, maka simpul dengan nilai terbesar **pasti** berada di sebelah kanan BST. Simpul dengan nilai terbesar dapat kita cari dengan merekursif BST terus menerus pada path sebelah kanan sampai bertemu dengan simpul tanpa memiliki anak kanan, yang berarti ialah simpul dengan nilai terbesar. Dari asumsi diatas, didapatkan konfigurasi basis dan rekurens:

Basis-0: IsTreeEmpty(BT) → false

Basis-1: not IsExistRight(BT) → F(Akar(BT)) {karena diminta menggunakan lambda}

Rekurens: IsExistRight(BT) → BSTMax(Right(BT), F

Sehingga, notasi fungsionalnya menjadi:

APAKAH MAX HABIS DIBAGI EMPAT? BSTMax(BT, F)

DEFINISI DAN SPESIFIKASI

BSTMax: BinarySearchTree, (integer → boolean) → boolean

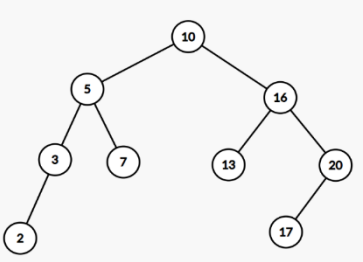
{BSTMax(BT, F) akan bernilai benar apabila nilai terbesar dalam BT habis dibagi 4.}

REALISASI

```
BSTMax(BT, F):
   depend on BT
      IsTreeEmpty(BT): false
      not IsExistRight(BT): F(Akar(BT))
      else: BSTMax(Right(BT), F)
         {bisa juga ditulis IsExistRight,
          namun saya rasa tidak perlu karena jika fungsi mencapai kondisi
          else, pasti node IsExistRight}
```

APLIKASI

Given a BinarySearchTree T:



→ BSTMax(T, λx.x mod 4 = 0)

OUTPUT: true

aplikasi nomor 4:

**import konstruktor dan selektor tree N-aire dari file yang pernah kalian buat dalam praktikum!*

```
def RTC(LPN, X):
   if isEmpty(LPN):
      return []
   else:
      return Konkat(SubTreeElements(FirstElmt(LPN), X), RTC(Tail(LPN), X))

def RTC2(LPN):
   if isEmpty(LPN):
      return []
   else:
      return Konkat(RENAT(FirstElmt(LPN)), RTC2(Tail(LPN)))

def RENAT(T):
   if isTreeNEmpty(T):
      return []
   else:
      return Konkat([akar(T)], RTC2(anak(T)))

def SubTreeElements(T, X):
   if isTreeNEmpty(T):
      return []
   else:
      if akar(T) == X:
         return RENAT(T)
      else:
         return RTC(anak(T), X)

T3 = makePN('K', [makePN('A', [makePN('Y', []), makePN('N', [makePN('S', [])])]),
makePN('F', [makePN('L', []), makePN('S', [])]), makePN('E', [makePN('I', [])])])

print(SubTreeElements(T3, 'A'))
```

aplikasi nomor 5:

**import konstruktor dan selektor binary tree dari file yang pernah kalian buat dalam praktikum!*

```
GivenATree = MakePB(10, MakePB(5, MakePB(3, [], []), MakePB(7, [], [])),
MakePB(17, MakePB(13, [], []), MakePB(20, [], [])))

def BSTMax(BT, F):
    if isTreeEmpty(BT):
        return []
    elif not IsExistRightPB(BT):
        return F(Akar(BT))
    else:
        return BSTMax(Right(BT), F)

print(BSTMax(GivenATree, lambda x: x % 4 == 0))
```