1 Programmer Følgende Funksjoner

1.1 `harEl:: (t -> Bool) -> [t] -> Bool`, slik at `harEl pr xs = True` hvis listen `xs` har et element `x` som tilfredsstiller predikatet `pr`, dvs, `pr x = True` og `False` ellers. F.eks:

```
harEl (==3) [1,1,2,3,2] = True
harEl (<3) [1,1,2,3,2] = True
harEl (>5) [1,1,2,3,2] = False
```

Svar. Bruker `any`.

```
harEl :: (t -> Bool) -> [t] -> Bool
harEl = any
```

1.1 `el :: (t -> Bool) -> [t] -> t` slik at `el pr xs` returnerer det første elementet `x` fra listen `xs` som tilfredsstiller predikatet `pr`. Funksjonen antar at et slikt element finnes i listen. F.eks:

```
el ((=='a').fst) [('b',2), ('a',3),('a',4)] = ('a',3)
el ((>3).snd) [('b',2), ('a',3), ('a',4)] = ('a',4)
```

Svar. Bruker `head` og `filter`.

```
el :: (t -> Bool) -> [t] -> t
el pr = head . filter pr
```

1.3 `gRep :: (t -> Bool) -> t -> [t] -> [t]` slik at `gRep pr y xs` erstatter med `y`, ethvert element `x` fra listen `xs` som tilfredsstiller predikatet `pr`. F. eks:

```
gRep (<'d') 'z' "abcd" = "zzzd"
gRep (=='a') 'x' "abcbcac" = "xbcbcxc"
```

Svar. Bruker `map` og en lambda funksjon

```
gRep :: (t -> Bool) -> t -> [t] -> [t]
gRep pr y = map (\x ->
  case () of
  _ | pr x -> y
    | otherwise -> x
  )
```

1.4 Vi bruker Binære trær med heltall lagret i alle noder (inklusivt blader) definert ved `data BT = B Int | N BT Int BT`. Programmer følgende funksjoner:

a) `elf :: BT -> Int -> Bool`, slik at `elt tr x = True` hvis tallet `x` forekommer i treet `tr`, og `False` ellers. F.eks `elt (N (B 1) 3 (B 0)) 2 = True` of `elt (B 1) 2 = False`.

b) `toL :: BT -> [Int]` slik at `toL tr` er en liste med alle tall som forekommer i i treet `tr`.

c) `dup :: BT -> Bool` slik at `dup tr = True` hvis noen tall forekommer (minst) to ganger i treet `tr` og `False` ellers.

Svar. a) Bruker rekursjon, pattern matching og guards

```
elt :: BT -> Int -> Bool
elt (B val) x = val == x
elt (N left val right ) x
  | val == x = True
  | val /= x = elt left x || elt right x
```

b) Bruker pattern matching

```
toL :: BT -> [Int]
toL (B x) = [x]
toL (N left val right) = toL left ++ [val] ++ toL right
```

c) Bruker en hjelpefunksjon

```
dup :: BT -> Bool
dup = dupL . toL

dupL :: [Int] -> Bool
dupL [] = False
dupL (x:xs) = elem x xs || dupL xs
```

2 Rettede Grafer

2.1 Programmer en funksjon `naboL :: Eq t => [(t,t)]->[(t,[t])] s` som konverterer en kantliste representasjon til en naboliste