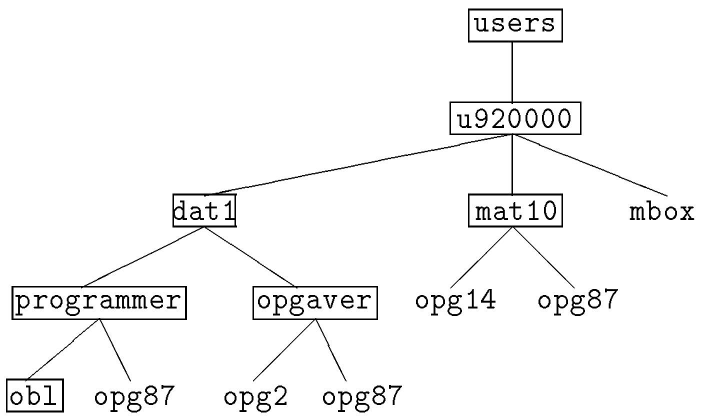
Opgave 1 (20%)

Som i TRINE noten side 186 kan et UNIX filsystem skitseres med følgende typer:

> **Type** FileList = **List** (File)
> **Type** File = **Prod** (name, data: Text)
> **Type** DirList = **List** (Directory)
> **Type** Directory = **Prod** (name: Text, files: FileList, dirs: DirList)

Det er behageligt at have en kommando, der kan finde alle forekomster af et givet filnavn. Betragt følgende værdi af type Directory:



Elementer i dirs tegnes med kasser og elementer i files tegnes uden. Filnavnet `opg87` har i dette filsystem tre forskellige forekomster, hvis fulde beskrivelser er:

```
/users/u920000/dat1/programmer/opg87
/users/u920000/dat1/opgaver/opg87
/users/u920000/mat10/opg87
```

> Skriv en TRINE procedure
>
> **Proc** FindFile[D: Directory] (f: Text)
>
> der udskriver de fulde beskrivelser af alle forekomster af filnavnet f i filsystemet D. Der lægges vægt på, at besvarelsen er letlæselig, detaljeret og korrekt. (Vink: lav en lokal rekursiv procedure med en ekstra parameter, der angiver vejen til D's rod.)

Opgave 2 (15%)

Et *trekantet* ligningssystem med n variabler er af formen:

$$\begin{aligned} x_0 &= k_0 \\ x_1 &= k_1 + a_{10}x_0 \\ x_2 &= k_2 + a_{20}x_0 + a_{21}x_1 \\ &\vdots \\ x_{n-1} &= k_{n-1} + a_{(n-1)0}x_0 + \cdots + a_{(n-1)(n-2)}x_{n-2} \end{aligned}$$

I TRINE kan koefficienterne til et trekantet ligningssystem repræsenteres som værdier af variablerne:

Var k: **List**(Int)
Var a: **List**(**List**(Int))

hvor tanken er, at k.(i) indeholder k_i og a.(i).(j) indeholder a_{ij}. At k og a indeholder koefficienterne til et trekantet ligningsystem, udtrykkes af prædikatet:

$$\textit{\textbf{trekantet}}(k,a):\quad (|k| = |a|) \;\wedge\; |a.(i)| = i$$

At x indeholder en løsning til ligningssystemet, udtrykkes af prædikatet:

$$\textit{\textbf{løsning}}(x,k,a):\quad (|x| = |a|) \;\wedge\; (\forall i \in 0..|x| :\; x_i = k_i + \sum_{j=0}^{i-1} a_{ij}x_j)$$

Betragt nedenstående algoritme.

Algoritme: Trekantløser
Stimulans: k,a: ***trekantet***(k,a)
Respons: x: ***løsning***(x,k,a)
Metode: x, i := **List**(?-**Int**| |a|), 0
do { ***løsning***(x(0..i), k(0..i), a(0..i)) $\wedge$ (0 $\leq$ i $\leq$ |k|) }
  i $\neq$ | k | $\rightarrow$
   ≪iterer≫
   i := i+1
od

Gør algoritmen færdig og bevis, at den er korrekt.

Opgave 3 (15%)

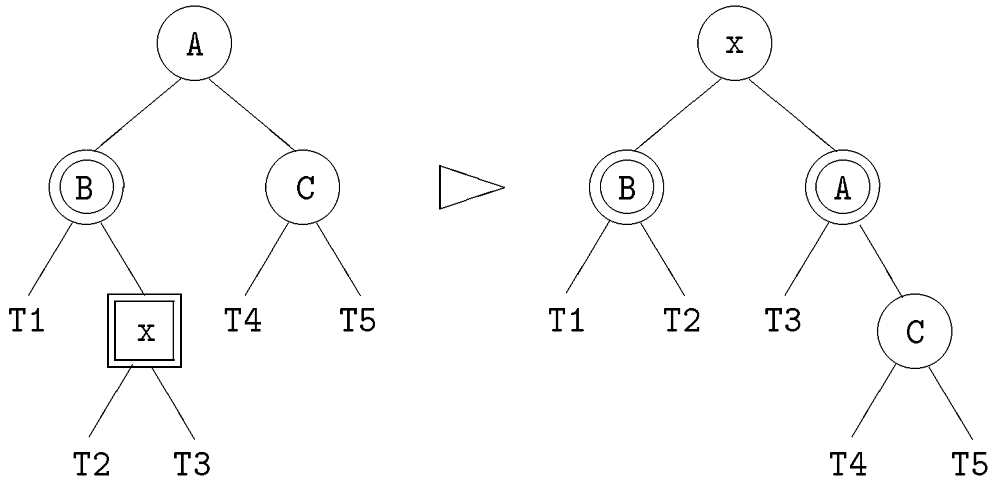
Denne opgave omhandler en udvidelse af de sædvanlige søgetræer, hvor vi til hver knude knytter oplysningen om summen af alle værdierne i det undertræ, som knuden er rod i.

a) Skitsér, hvordan de sædvanlige søgetræsoperationer **Insert** og **Delete** kan udvides, så de ekstra oplysninger vedligeholdes og udførelsestiderne stadig er proportionale med træets højde.

Vi er interesserede i en ekstra operation **Add**$[D](k)$, der beregner summen af de elementer i søgetræet D, der er mindre end eller lig med k.

b) Giv en formel specifikation af operationen **Add** og skitsér, hvordan den i et udvidet søgetræ kan implementeres, så udførelsestiden er proportional med træets højde.

Som altid, er vi interesserede i at holde søgetræerne balancerede. Det kan fx gøres ved hjælp af den rød-sorte teknik, der benytter sig af rotationer som den følgende:



Man kan dog ikke umiddelbart transformere et udvidet søgetræ på denne måde.

c) Vis med ovenstående rotation som eksempel, hvordan den rød-sorte teknik skal tilpasses vores udvidede søgetræer.

Opgave 4 (15%)

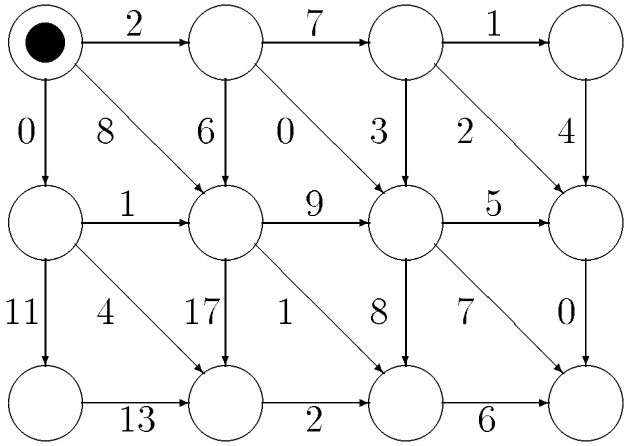
Betragt følgende TRINE program.

```
      (+ Box S
           Type T = Int

           Proc P(m: T, n: Int)
              skip
           end P
         end S

         Type A = Prod(x: C, y: B)
         Type B = List(C)
         Type C = Sum(z: A, i: Int)
         Type D = Prod(x: C, y: B)

         Var a: A
         Var b: B
         Var c: C
         Var d: D

(*1*)    a := d
(*2*)    c := Sum(1: Prod(c, b))
(*3*)    S'P(87, d.y.(1).i)
(*4*)    d := Prod(?-Sum, ?-List)
      +)
```

> Angiv for hver af sætningerne (*1*), (*2*), (*3*) eller (*4*) om de vil give anledning til fejl fra TRINE oversætteren? Begrund dine svar.

Opgave 5 (20%)

Et ***skråplan*** er en orienteret vægtet graf, med ikke-negative vægte, i hvilken knuderne sidder i et rektangulært gitter, og hver knude har en kant til dens eventuelle sydlige, østlige og sydøstlige nabo. Den nordvestligste knude i grafen kaldes for dens *rod*. Det følgende er et eksempel med 12 knuder, hvor roden er farvet sort.



a) Hvis et skråplan har r rækker og s søjler, så har det naturligvis $n = r \cdot s$ knuder. Hvor mange kanter har det, udtrykt ved r og s?

Vi antager i det følgende en passende repræsentation af et skråplan, der tillader, at man i konstant tid kan komme frem og tilbage mellem en knude og dens sydlige, østlige og sydøstlige naboer.

Vi er interesserede i at finde længden af den korteste vej fra roden til hver af de øvrige knuder.

b) Hvad er tidskompleksisteten for at beregne dette for et skråplan med n knuder, hvis vi benytter Dijkstras algoritme?

c) Beskriv en algoritme, der løser problemet i tid $O(n)$.

Vi er også interessede i for hvert par af knuder at finde længden af den korteste vej mellem dem.

d) Hvad er tidskompleksiteten for at beregne dette for et skråplan med n knuder, hvis vi benytter Floyds algoritme?

e) Beskriv en algoritme, der løser problemet i tid $O(n^2)$.

Opgave 6 (15%)

Betragt nedenstående grammatik.

$$
\begin{array}{lll}
\mathbf{A} & ::= & \texttt{a} \mid \\
 & & \texttt{x}\mathbf{A} \mid \\
 & & \mathbf{A}\texttt{yx} \mid \\
 & & \mathbf{AA}
\end{array}
$$

Følgende procedure afgør, om t(i..j) kan genereres af denne grammatik.

```
Proc A[t: Text](i, j: Int) → (Bool)
    if (i+1=j) ∧ (t.(i) = 'a') → return true fi
    if (i+1<j) ∧ (t.(i) = 'x') ∧ A[t](i+1, j) → return true fi
    if (i+2<j) ∧ (t.(j-2) = 'y') ∧ (t.(j-1) = 'x') ∧ A[t](i, j-2) →
        return true
    fi
    (+ Var k: Int
        k:=i+1
        do k<j →
            if A[t](i, k) ∧ A[t](k, j) → return true fi
            k:=k+1
        od
    +)
    return false
end A
```

a) Forklar kort i ord, hvorledes proceduren virker.

Man kan vise, at $T[\![\textbf{proc}\ \mathrm{A}]\!](n) \in \Omega(2^n)$, hvor n er længden af teksten t.

b) Forklar, hvordan man kan benytte dynamisk programmering til at forbedre proceduren. Hvad bliver den forbedrede tidskompleksitet?