Opgave 1 (20%)

Et polynomium i variablen x kan beskrives med følgende typer Expr og Poly:

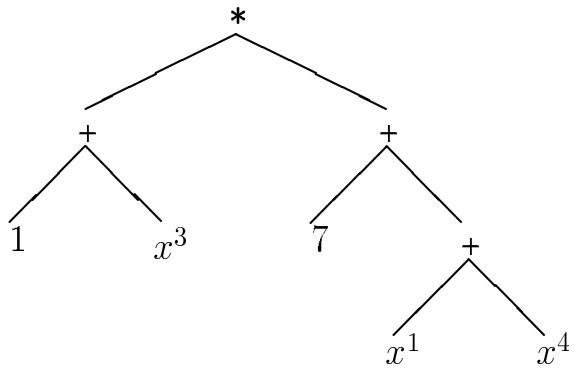
Type Expr = **Sum**(const: Int, x: Int, plus, times: Args)
Type Args = **Prod**(left, right: Expr)

Type Poly = **List**(Int)

Vi har således, at Expr-udtrykket:

```
Expr(times:Args(Expr(plus: Args(Expr(const: 1), Expr(x: 3))),
                Expr(plus: Args(Expr(const: 7),
                                Expr(plus: Args(Expr(x: 1), Expr(x: 4)))
                                )
                     )
                )
     )
```

svarer til "værdien":



der beskriver polynomiet:

$$(1 + x^3)(7 + (x^1 + x^4))$$

der kan reduceres til den sædvanlige form:

$$(7 + x^1 + 7x^3 + 2x^4 + x^7)$$

hvilket endeligt svarer til Poly-værdien:

$$(7, 1, 0, 7, 2, 0, 0, 1)$$

Typen Expr beskriver således polynomier som udtryk, hvorimod typen Poly beskriver polynomier i form af deres koefficienter.

Antag i det følgende eksistensen af en værdiprocedure:

Proc Mult(p, q: Poly) → (Poly)

der returnerer produktet af de to polynomier p og q.

Skriv en TRINE værdiprocedure:

Proc Eval(e: Expr) → (Poly)

der oversætter en Expr-værdi til den tilsvarende Poly-værdi. Der lægges vægt på, at besvarelsen er letlæselig, detaljeret og korrekt.

Opgave 2 (20%)

Den foregående opgave forudsatte eksistensen af en værdiprocedure, der kan beregne produktet af to polynomier, repræsenterede som værdier af følgende type

Type Poly = **List**(Int)

hvor den i'te indgang angiver koefficienten af x^i.

Bemærk, at vi tillader *foranstillede 0'er* i en Poly-værdi. Værdien:

$$(7, 1, 0, 7, 2, 0, 0, 1, 0, 0, 0)$$

er således også en repræsentation af polynomiet fra opgave 1.

I denne opgave skal vi vise korrekthed af en algoritme, der udfører denne beregning. I det følgende lader vi $p \oplus q$ angive summen og $p \otimes q$ produktet af polynomierne p og q. Betragt nu algoritmen:

```
Algoritme: Polynomiemultiplikation
Stimulans: p,q: Poly
Respons:   r: r=p⊗q
Metode:    r:=Poly(0||p|+|q|-1)
           i:=0
           do {I1}
              i<|p| → j:=0
                      do {I2}
                         j<|q| → r.(i+j):=r.(i+j)+p.(i)*q.(j)
                                 j:=j+1
                      od
                      i:=i+1
           od
```

Invarianten for den indre **do**-løkke er:

$$I_2\text{: } r = (p(0..i) \otimes q) \oplus (\text{Poly}(0|i) ++ \text{Poly}(p.(i))) \otimes q(0..j)$$

a) Forklar i ord, hvad denne invariant betyder.

I det følgende kan det uden bevis benyttes, at I_2 er gyldig.

b) Angiv en passende invariant $\{I_1\}$ og bevis algoritmens gyldighed og korrekthed.

Opgave 3 (20%)

Denne opgave omhandler forskellige udvidelser til de rød-sorte søgetræer.

Som bekendt er den sorte højde af en knude lig med det største antal sorte knuder på en vej fra knuden til et blad. Vi er interesserede i at udvide de rød-sorte søgetræer, således at hver knude indeholder et felt, der angiver dens sorte højde.

a) Forklar kort, hvorledes informationen om den sorte højde kan vedligeholdes under de sædvanlige operationer på rød-sorte søgetræer uden at forøge den asymptotiske tidskompleksitet.

I det følgende antager vi således, at vi benytter denne udvidelse. Vi er nu interesserede i at implementere en operation

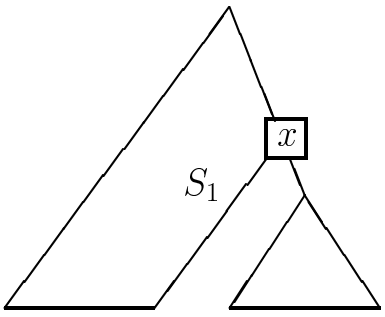
$$\text{Join}(S_1, e, S_2)$$

hvor S_1 og S_2 er (udvidede) rød-sorte søgetræer og e er et element, hvorom det gælder, at $S_1 < e < S_2$ (det vil sige, alle elementer i S_1 er mindre end e, der er mindre end alle elementer i S_2) og at $|S_1| > |S_2|$ (det vil sige, S_1 indeholder flere elementer end S_2).

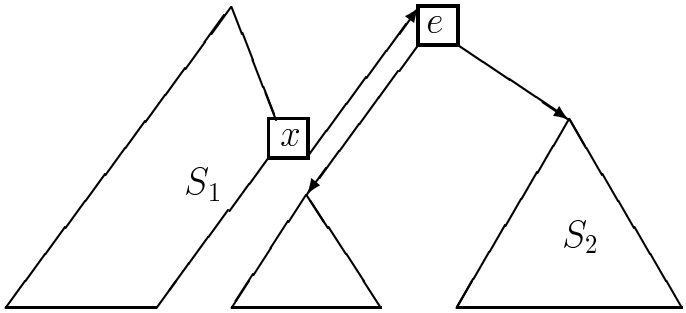
Join skal returnere et rødt-sort søgetræ med elementerne $S_1 \cup \{e\} \cup S_2$. Operationen tillades at være destruktiv, hvilket betyder, at S_1 og S_2 gerne må ødelægges under udførelsen.

b) Forklar, hvorledes man med lethed kan implementere Join i tid $O(|S_2| \log |S_1|)$.

Lad nu x være en vilkårlig knude på vejen fra roden af S_1 til dets største element (der jo ligger længst til højre):



Betragt derefter følgende sammensyning:



c) Argumentér for, at dette altid resulterer i et søgetræ (men ikke nødvendigvis i et rødt-sort søgetræ).

d) Argumentér for, at man med udgangspunkt i at vælge et passende x, kan implementere Join i tid $O(\log |S_1|)$.

Opgave 4 (20%)

RASMUS-relationen L indeholder informationer om kurser og lærere:

kursus:Text	lærer:Text
Dat 1	Erik
Dat 1	Michael
Mat 10	Jørgen
Mat 11	Ebbe
Mat 11	Tage
⋮	⋮

Vi antager, at alle lærere har indbyrdes forskellige navne, men at en lærer godt kan undervise flere kurser på samme tid.

RASMUS-relationen S indeholder information om studenter og deres eksamensforsøg:

årskort:Text	kursus:Text	karakter:Int
940001	Dat 1	8
940001	Mat 10	5
940002	Mat 11	7
940087	Dat1	13
⋮	⋮	⋮

a) Skriv et RASMUS-udtryk, der angiver alle de kurser, hvor der er mindst to lærere, der underviser.

Følgende udtryk søger alle at beregne en relation, der angiver alle de lærere, som studenten med årskort k har haft.

1) `((L*S)?(#.årskort=k))|+lærer`
2) `(L*(S?(#.årskort=k)))|+lærer`
3) `(L|+lærer)*(S?(#.årskort=k))`

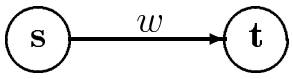
b) Hvilke af udtrykkene er korrekte, og hvilket korrekt udtryk er mest effektivt? Begrund dine svar.

c) Skriv et RASMUS-udtryk, der for hver lærer angiver antallet af studenter, som har bestået mindst et af de kurser, læreren underviser på.

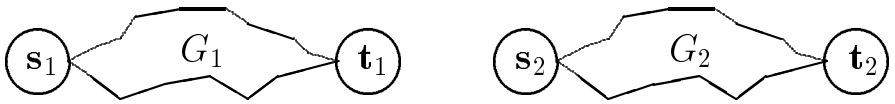
Opgave 5 (20%)

Denne opgave omhandler *flade grafer*, der er specielle orienterede, vægtede grafer. De har altid to bestemte knuder **s** og **t**, og defineres i øvrigt induktivt som følger.

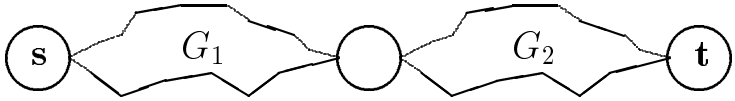
Den mindste flade graf består blot af to knuder og en enkelt kant:



To flade grafer:

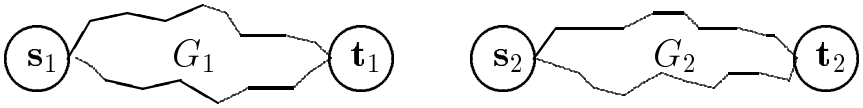


kan danne en ny ved at blive sat sammen i *serie*:

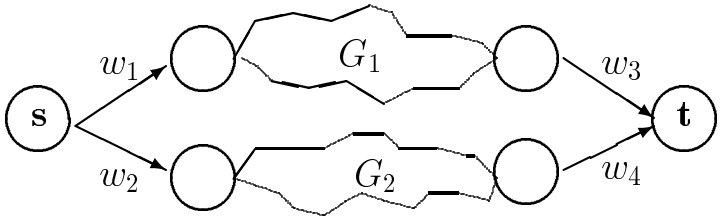


Her sammensmelter man **t**-knuden for G_1 og **s**-knuden for G_2 til en almindelig knude.

To flade grafer:



kan danne en ny ved at blive sat sammen i *parallel*:



Her introducerer man to nye knuder, der bliver henholdsvis **s**-knuden og **t**-knuden, og fire nye kanter.

a) Lad $k(n)$ være antallet af kanter i en flad graf med n knuder. Argumentér for, at $k(n) \in \Theta(n)$.

For en flad graf er vi nu interesserede i at finde længden af den korteste vej fra **s**-knuden til **t**-knuden.

b) Hvor lang tid vil det med Dijkstras algoritme tage at løse denne opgave for en flad graf med n knuder?

c) Hvordan kan man udnytte den specielle struktur af flade grafer til at opnå en mere effektiv algoritme?