Opgave 1 (25%)

Følgende grammatik

```
ML ::= ML ML
     | <b> ML </b>
     | <i> ML </i>
     | text
```

definererer et simpelt markeringssprog, der tillader at man i en tekst specificerer anvendelse af fed skrift med `<b>` … `</b>` og anvendelse af kursiv skrift med `<i>` … `</i>`. F.eks. skal den markerede tekst

```
<b> Quote from <i> Peter M. Senge </i> </b>
As one seasoned organization consultant once put it,
<i> People don’t resist change; they resist being changed </i>
```

udskrives således:

> **Quote from *Peter M. Senge***
>
> As one seasoned organization consultant once put it,
>
> *People don’t resist change; they resist being changed.*

En markeret tekst kan repræsenteres ved hjælp af en række Java klasser

```
abstract class ML {}
class Composite extends ML { ML[] mark_up_texts; }
class Bold extends ML { ML mark_up_text; }
class Italic extends ML { ML mark_up_text; }
class Text extends ML { String text; }
```

a) Tilføj en metode

```
boolean nestedMarkup() { ...  }
```

til klassen ML og dens subklasser, så et kald af metoden returnerer en sandhedsværdi, der fortæller om en markering ... eller <i>...</i> forekommer inden i en anden markering (ligeledes ... eller <i>...</i>). Bemærk, at det kan være hensigtsmæssigt at definere en ekstra hjælpe-metode i alle klasser. Der lægges vægt på at besvarelsen er letlæselig, detaljeret og korrekt.

Vi vedtager nu følgende konvention for udskrivning af markerede tekster: Hvis ... forekommer inden i en anden ..., så ignoreres den inderste markering, hvorimod en forekomst af <i>...</i> inden i en anden <i>...</i> skal bevirke at vi undlader at anvende kursiv skrift, dvs at en tekst udskrives med kursiv netop når den er omgivet af et ulige antal <i>...</i>. F.eks. skal følgende markerede tekst

```
Datalogi <i>er ganske <i>overordentligt <b>morsomt,
<i>hvilket <b>jeg </b>gerne </i></b>står </i>ved.</i>
```

udskrives således:

Datalogi *er ganske* overordentligt **morsomt,**
hvilket jeg gerne står *ved.*

Til hjælp ved udskrivningen er defineret følgende 5 metoder:

```
void beginItalic() {...}
void endItalic() {...}
void beginBold() {...}
void endBold() {...}
void printText(String s) {..}
```

F.eks vil den markerede tekst i det sidste eksempel blive udskrevet korrekt med følgende sekvens af metodekald

```
printText("Datalogi ")
beginItalic();
printText("er ganske ")
endItalic();
printText("overordentligt ")
beginBold();
printText("morsomt, ")
beginItalic();
printText("hvilket ")
printText("jeg ")
printText("gerne ")
endItalic();
endBold();
printText("står ")
beginItalic();
printText("ved.")
endItalic();
```

Bemærk, at intet `beginItalic...endItalic` par forekommer imellem et andet `beginItalic...endItalic` par, og tilsvarende at intet `beginBold...endBold` par forekommer imellem et andet `beginBold...endBold` par.

b) Tilføj en metode

`void udskriv(boolean italic, boolean bold) { ... }`

til klassen `ML` og dens subklasser, så et kald af metoden bevirker at den pågældende markerede tekst udskrives i overensstemmelse med værdien af parametrene `italic` og `bold`. Du kan frit kalde metoderne `beginItalic`, `endItalic`, `beginBold`, `endBold` og `printText`. Der lægges vægt på at besvarelsen er letlæselig, detaljeret og korrekt.

Opgave 2 (20%)

Følgende 5 intervaller

$$
\begin{array}{l}
[1, 2, 3, 4, 5] \\
[6, 7, 8, 9] \\
[10] \\
[11, 12] \\
[13, 14, 15, \ldots]
\end{array}
$$

er disjunkte og omfatter tilsammen samtlige naturlige tal, idet det sidste interval er uendeligt stort.

En *intervalopdeling* af de naturlige tal består af en sådan endelig mængde af ikke-tomme, disjunkte intervaller, der tilsammen omfatter alle naturlige tal.

En intervalopdeling kan naturligt repræsenteres ved sin *repræsentantmængde*, som består af det første tal i hvert interval. F.eks. har intervalopdelingen i eksemplet ovenfor følgende repræsentantmængde

$$I = \{1, 6, 10, 11, 13\}$$

Vi vil beskæftige os med den abstrakte datatype **NatInt**, der understøtter følgende operationer på en intervalopdeling af de naturlige tal:

> `lookup`(x): Returnerer repræsentanten for det interval, hvor tallet x ligger.
>
> `union`(x): Det interval, hvor x ligger, forenes med det efterfølgende interval (forudsat at x ikke ligger i det sidste interval).
>
> `split`(x): Det interval $[a, \ldots, b]$, hvor x ligger, opsplittes i to intervaller $[a, \ldots, x-1]$ og $[x, \ldots, b]$ (forudsat at x ikke er repræsentant for sit interval).

a) Angiv en implementation af **NatInt**, så alle tre operationer kan udføres i tid $O(\log n)$, hvor n er antal intervaller i intervalopdelingen. Vink: Hvad sker der med repræsentantmængden for en intervalopdeling, når der udføres en union eller en split?

Vi ser nu på den abstrakte datatype **NextNatInt**, der er ligesom **NatInt**, pånær at den understøtter en ekstra operation

> `findnext`(x, d): returnerer repræsentanten for det første interval af længde mindst d efter x's interval (forudsat at x ikke ligger i det sidste interval).

Hvis vi som eksempel betragter intervalopdelingen fra før, der kan repræsenteres ved

$$\{1, 6, 10, 11, 13\}$$

så vil `findnext`$(7, 2)$ returnere 11.

b) Angiv en implementation af den udvidede datatype **NextNatInt**, hvor alle de angivne operationer udføreres i tid $O(\log n)$, hvor n er antal intervaller i intervalopdelingen.

Opgave 3 (30%)

Denne opgaver omhandler *palindromer*. Et palindrom er som bekendt en tekststreng, der fremtræder ens, hvadenten den læses forfra eller bagfra.

Enhver tekststreng kan opfattes som sammensat af en række palindromer, hvis man tillader at et palindrom kun består af et enkelt bogstav. F.eks. kan vi skrive *Otto er reder* på bl.a. følgende måder, idet blanke ignoreres:

```
"ottoerreder" = "otto" + "erre" + "d" + "e" + "r"
"ottoerreder" = "otto" + "e" + "r" + "reder"
"ottoerreder" = "o" + "t" + "t" + "o" + "e" + "r" + "reder"
```

På en vilkårlig tekststreng s definerer vi følgende funktion:

> **MinPal**(s) er det mindste antal palindromer, hvorudfra man kan sammensætte strengen s; dvs. det er det mindste k, så s kan skrives på formen $s = w_1 w_2 \ldots w_k$, hvor alle $w_1, \ldots, w_k$ er palindromer.

For eksempel er **MinPal**("`ottoerreder`") = 4 fordi

```
"ottoerreder"  = "otto" + "e" + "r" + "reder"
```

og det er ikke muligt at skrive `"ottoerreder"` som en sammensætning af 3 eller færre palindromer.

a) Argumenter for at

$$\begin{aligned} &\mathbf{MinPal}(s[i..j-1]) \\ &\quad = \min_{i<k<j} \{\mathbf{MinPal}(s[i..k-1]) + \mathbf{MinPal}(s[k..j-1])\}. \end{aligned}$$

b) Angiv en effektiv dynamisk programmering algoritme til at beregne **MinPal**. Hvad bliver udførelsestiden?

På en vilkårlig tekststreng s definerer vi endnu en funktion:

> **MaxPal**(s) er længden af det længste palindrom, som er et prefix af s.

For eksempel har `"ottoerreder"` præcis to prefixer, der samtidig er palindromer, nemlig `"o"` og `"otto"`. Heraf er `"otto"` det længste og derfor er **MaxPal**(”`ottoerreder`”) = 4.

Definer funktionen *reverse*(s) til at være strengen s skrevet bagfra, f.eks. er *reverse*(”`alabast`”) = ”`tsabala`”.

Betragt nu følgende figur:

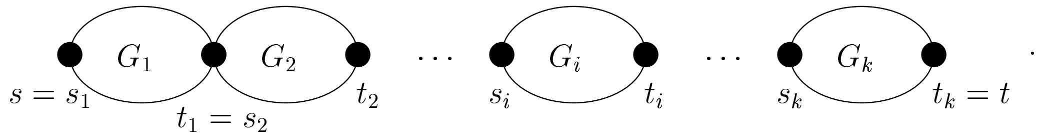
reverse(s):	t	s	a	b	a	l	a				
s:					a	l	a	b	a	s	t

c) Argumentér for at **MaxPal**(s) er længden af det længste prefix af s, der forekommer som et suffix af *reverse*(s).

d) Forklar, hvordan man kan modificere Knuth-Morris-Pratts mønstergenkendelsesalgoritme, så den anvendt på teksten $T =$*reverse*(s) og mønstret $P = s$ beregner **MaxPal**(s). Hvad bliver tidskompleksiteten af den resulterende algoritme?

Opgave 4 (25%)

$G_1, \ldots, G_k$ er orienterede vægtede grafer med ikke-negative vægte. G_i har en startknude s_i og en slutknude t_i. Betragt følgende graf G, som fremkommer ved at identificere t_i med s_{i+1} og ved at lade s_1 og t_k være start- og slutknuder s og t i G.



a) En effektiv algoritme til at finde korteste vej fra s til t består i at udføre Dijkstras algoritme på hver af G_i'erne og returnere summen af resultaterne. Angiv denne algoritmes udførelsestid (når G_i indeholder n_i knuder og m_i kanter).

Lad n (m) være antallet af knuder (kanter) i G. Antag at graferne G_i er "lige store", dvs at $n_i = n_j$ og $m_i = m_j$ for $i, j = 1, \ldots, k$.

b) Angiv udførelsestiden fra **a**) udtrykt ved n, m og k. For hvilke af følgende k-værdier er den bedre end udførelsestiden for Dijkstras algoritme udført direkte på G? (Argumenter for dit svar)

- k er $\Theta(1)$
- k er $\Theta(\sqrt{n})$
- k er $\Theta(n)$

Antag nu, at der er en vis fleksibilitet mht. valg af start- og slutknuder s_i og t_i. Mere præcist er der for hver graf G_i angivet to mængder af knuder S_i og T_i, blandt hvilke start- henholdsvis slutknuder skal findes.

c) Angiv en effektiv algoritme, der givet G_i, S_i og T_i ($i = 1, \ldots, k$) finder start- og slutknuder (s_i, t_i), $i = 1, \ldots, k$ så den korteste vej fra s til t bliver kortest mulig.

d) Antag at ikke alene G_i'erne, men også S_i'erne og T_i'erne, er lige store, dvs. $|S_1| = \cdots = |S_n| = |T_1| = \cdots = |T_n| = d$. Angiv udførelsestiden for algoritmen fra **c)** udtrykt ved n, m, k og d.