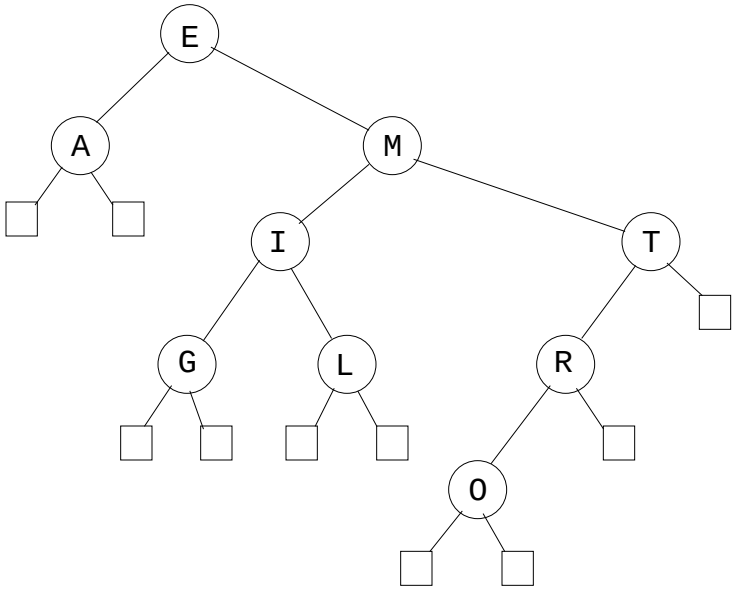
Vejledende løsninger til opgave 1 og 2

Opgave 1: Binære søgetræer

Spørgsmål 1.1

```
BinarySearchTree bst = new BinarySearchTree();
bst.insert('E');
bst.insert('M');
bst.insert('T');
bst.insert('I');
bst.insert('R');
bst.insert('O');
bst.insert('G');
bst.insert('L');
bst.insert('A');
```

Spørgsmål 1.2



Kommentar: Det er ikke et krav, at de eksterne knuder medtages (kvadraterne på tegningen). Blot venstre-højre-orienteringen af træets grene fremgår tydeligt.

Spørgsmål 1.3

Rekursiv udgave:

```
private boolean contains(char key, Node t) {
    if (t == null)
        return false;
    if (key < t.key)
        return contains(key, t.left);
    if (key > t.key)
        return contains(key, t.right);
    return true;
}
```

Ikke-rekursiv udgave:

```
private boolean contains(char key, Node t) {
    while (t != null && key != t.key)
        t = key < t.key ? t.left : t.right;
    return t != null;
}
```

Kommentar: Metoden kan programmers på mange andre måder end de her anførte.

Spørgsmål 1.4

Med ord:

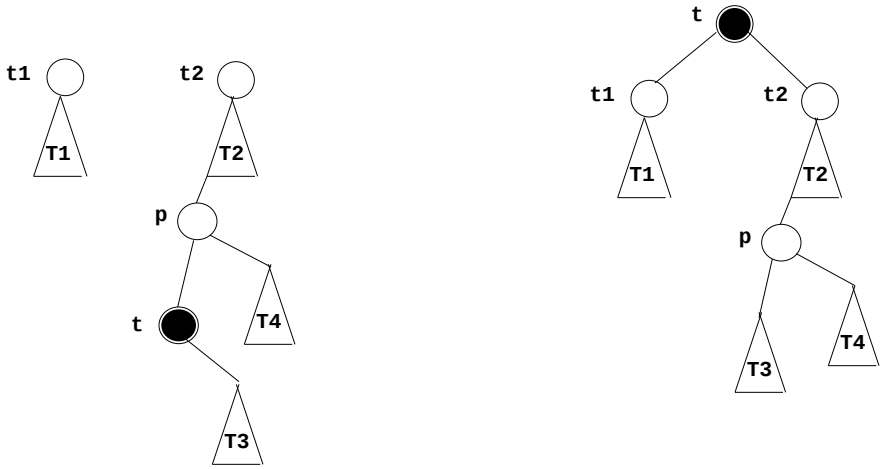
For den knude i et det binære søgetræ, der skal slettes, sammensættes dens venstre undertræ med dens højre undertræ, således at resultatet bliver et binært søgetræ.

Dette opnås ved at flytte den knude t, der er længst til venstre i det højre undertræ, op som rod. Som venstre undertræ for roden benyttes det oprindelige venstre undertræ. Som rodens højre undertræ anvendes det oprindelige højre undertræ, idet t dog er fjernet.

Hvis det oprindelige højre undertræ imidlertid er tomt, bliver resultat af sammensætningen blot det venstre undertræ.

Metodens returværdi er roden for det resulterende binære søgetræ.

Med en figur:



Spørgsmål 1.5

Metoden kan f.eks. programmeres som en rekursiv symmetrisk gennemgang (*inorder traversal*) af træet.

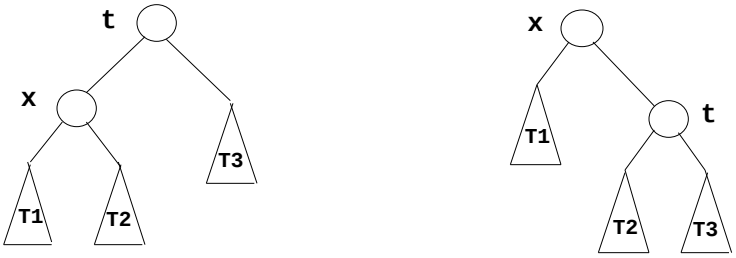
```
public void printSorted()
  { printSorted(root); System.out.println(); }

private void printSorted(Node t) {
    if (t == null) return;
    printSorted(t.left);
    System.out.print(t.key + " ");
    printSorted(t.right);
}
```

Kommentar: Det centrale i opgavebesvarelsen er, at knuderne besøges i den korrekte rækkefølge.

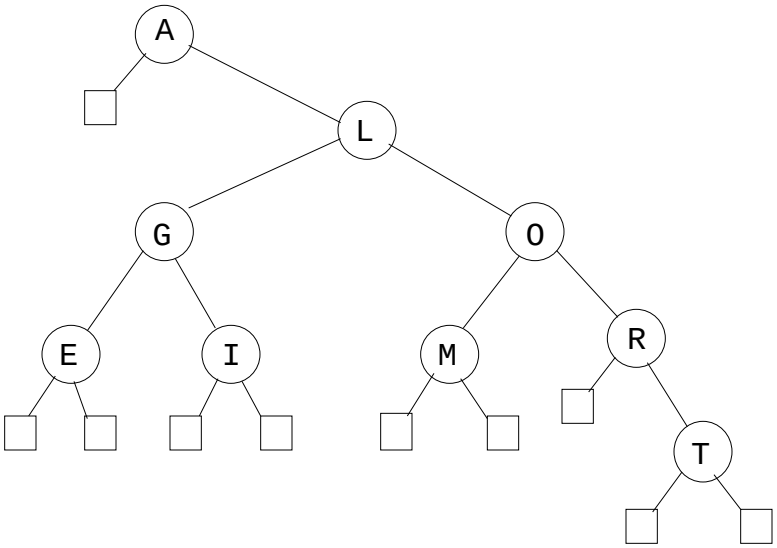
Spørgsmål 1.6

```
Node rotateRight(Node t) {
    Node x = t.left;
    t.left = x.right;
    x.right = t;
    return x;
}
```

Kommentar: I forhold til metode `rotateLeft` *er* `left` *og* `right` *blot ombyttet.*



Spørgsmål 1.7



Kommentar: Rotationerne i `insert` *bevirker, at en knude altid indsættes som ny rod i det binære søgetræ.*

Opgave 2: Minimale udspændende træer

Spørgsmål 2.1

Nabomatrixrepræsentation:

	A	B	C	D	E	F	G
A	0	2	7	1	0	0	0
B	2	0	0	3	10	0	0
C	7	0	0	8	0	9	0
D	1	3	8	0	6	10	11
E	0	10	0	6	0	0	5
F	0	0	9	10	0	0	12
G	0	0	0	11	0	12	0

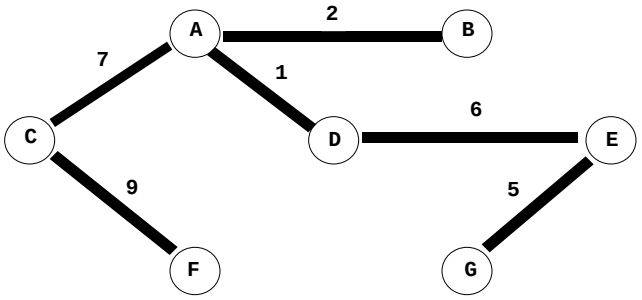
Kommentar. Diagonalelementernes værdier er uden betydning. Ikke-eksisterende kanter repræsenteres her ved værdien 0, men det er også i orden at repræsentere dem ved meget store værdier.

Nabolisterepræsentation:

A → B 2 → C 7 → D 1 → □
B → A 2 → D 3 → E 10 → □
C → A 7 → D 8 → F 9 → □
D → A 1 → B 3 → C 8 → E 6 → F 10 → G 11 → □
E → B 10 → D 6 → G 5 → □
F → C 9 → D 10 → G 12 → □
G → D 11 → E 5 → F 12 → □

Kommentar. Knudernes rækkefølge i nabolisterne er ligegyldig.

Spørgsmål 2.2



Spørgsmål 2.3:

(A,D), (A,B), (D,E), (E,G), (A,C), (C,F)

Spørgsmål 2.4

(A,D), (A,B), (E,G), (D,E), (A,C), (C,F)