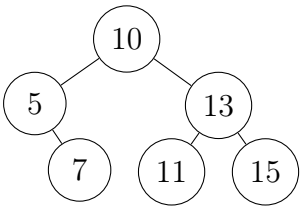
Uppgift 2: Binära sökträd

Inom databehandling används olika datastrukturer för att lagra data, så att användningen av data är så effektiv som möjligt. En central datastruktur heter *binärt sökträd*. Ett binärt sökträd består av en mängd *noder* som är kopplade ihop. Noderna innehåller nyckelvärden. Ofta lagras också diverse nyttig information i noderna. Det är denna information som man söker ur trädet. I denna uppgift lagrar vi endast nyckelvärden.



Ett enkelt binärt sökträd.

Datastrukturens noder, liksom själva trädet, har *egenskaper* som beskriver dem. Om noden x har egenskapen z, görs referens till värdet för denna egenskap med beteckningen z[x]. Vi betecknar med värdet NIL situationen där x inte har egenskapen ifråga (egenskap[x]=NIL).

Binärträdets nod x har följande egenskaper:

- key[x]: nyckelvärdet för noden x (nyckelvärdet är ett heltal)
- left[x]: referens till det vänstra barnet för noden x (en nod eller NIL)
- right[x]: referens till det högra barnet för noden x (en nod eller NIL)
- parent[x]: referens till föräldern för noden x (en nod eller NIL)

Det binära sökträdet P har följande egenskap:

- root[P]: referens till trädets rot (en nod eller NIL)

Vi antar i denna uppgift, att inga nyckelvärden finns flera gånger i trädet. Beteckningen left[x]=NIL betyder att noden x saknar vänster barn. Sådana noder som saknar båda barnen kallas *lövnoder*. En nods *avkomma* är alla dess barn, dessas barn osv ända till lövnoderna. Nodens *anfäder* är dess förälder, dennas förälder osv. ända till roten. Det *vänstra delträdet* för en nod x innehåller noden left[x] och alla dess avkommor (delträdet kan vara tomt). På motsvarande sätt definieras det högra delträdet för en nod.

Följande krav gäller för ett binärt sökträd:

- Om root[P]=NIL, så är trädet P tomt.
- Rotnoden har ingen förälder (dvs parent[root[P]] = NIL). Alla andra noder i trädet har en förälder.
- Varje nod har maximalt två barn, det vänstra och det högra. Endera av dem eller båda kan saknas.
- Låt x vara en godtycklig nod i trädet. Om y är en nod i dess vänstra delträd, så är key[y] $<$ key[x]. På motsvarande sätt, om y finns i det högra delträdet till x, så är key[y] $>$ key[x].

Ett binärt träd kan åskådliggöras genom att rita det så att noderna är cirklar inom vilka nyckelvärdena finns och var förälder kopplas ihop med sina barn med ett streck. Brukligt är att placera noderna i figuren på detta sätt:

- Rotnoden placeras överst.
- Rotnodens barn placeras så att left[root[P]] placeras ner till vänster om roten och right[root[P]] ner till höger. Dessa barnnoder placeras sinsemellan på samma höjd. Om endera av barnen saknas, ritas det naturligtvis inte.
- Barnnodernas barn och senare efterföljare ritas på motsvarande sätt neråt, så att noderna på lika många stegs avstånd från roten placeras på samma höjd.

I det föjande betyder uttrycket *nod a* noden med nyckelvärdet a. I exemplet i början av uppgiften gäller följande egenskaper: root[P] är noden 10. Vänster barn till noden 10 är noden 5 och höger barn är noden 13. Föräldern till noden 5 är noden 10. Noden 5 saknar det vänstra barnet.

En *algoritm* är en beskrivning över hur man skall göra för att utföra en given uppgift. Uppgiften har som parametrar *input*, som algoritmen bearbetar. Inputvärdena noteras inom parentes efter algoritmens namn. Vi behandlar två slag av algoritmer: sådana som beräknar något *resultat* från det binära trädet (till exempel det största nyckelvärdet som finns i trädet), och sådana som ändrar strukturen på det binära trädet (till exempel lägger till en nod). Algoritmen presenteras med en algoritmnotation där man kombinerar enskilda instruktioner som utförs i ordning en efter en, som utförs i enlighet med något villkor, eller som utförs flera gånger. För att tillfälligt lagra information används *variabler*, som kan innehålla till exempel tal, referenser till en nod eller ett träd, eller ha värdet NIL. Värdet på en variabel tilldelas eller ändras med en *tilldelningssats*. Exempelvis ger instruktionen **tilldela** x ← 3 talet 3 som värde för variabeln x. Om därefter instruktionen **tilldela** x ← 5 utförs, så ersätts det tidigare värdet med talet 5.

I det som följer presenterar vi några centrala algoritmer för behandlingen av binära träd. Algoritmen SearchTree får som input trädet och ett tal och ger som resultat den nod som har detta som sitt nyckelvärde (eller värdet NIL, om en sådan nod inte existerar). AddNode lägger en ny nod till trädet. MinValue söker trädets nod med det minsta nyckelvärdet (eller värdet NIL för ett tomt träd).

SearchTree(träd P, nyckelvärde a)

1. **tilldela** x ← root[P]
2. **så länge som** x≠NIL **och** key[x]≠a **utför raderna** 3-4
3. **om** a<key[x] **så tilldela** x ← left[x]
4. **annars tilldela** x ← right[x]
5. **resultatet är** x

Algoritmen SearchTree får som input trädet P som behandlas och nyckelvärdet a som söks ur trädet. På rad 1 tilldelas variabeln x en referens till trädets rot. Därefter kontrollerar algoritmen ett villkor på rad två. Om x har värdet NIL, är trädet tomt; motsvarande om x har nyckelvärdet a, har värdet hittats. I båda fallen utförs inte raderna 3-4 en enda gång. Annars går algoritmen vidare till raderna 3 och 4 och i varje omgång går den vidare antingen till nodens vänstra eller högra delträd, på basen av en jämförelse mellan det sökta värdet och nyckelvärdet i noden. Då algoritmen avslutas är resultatet x antingen noden som innehåller det sökta värdet eller NIL, om det sökta värdet inte alls finns i trädet.

Som nästa algoritm presenterar vi tillägg av en nod till trädet. Vi antar att noden s som läggs till har följande utgångsvärden på sina egenskaper:

- key[s] = något heltal
- left[s] = NIL, right[s] = NIL, parent[s] = NIL

AddNode(träd P, nod s)

1. **tilldela** y ← NIL
2. **tilldela** x ← root[P]
3. **så länge som** x≠NIL **utför raderna** 4-6
4. **tilldela** y ← x
5. **om** key[s]<key[x] **så tilldela** x ← left[x]
6. **annars tilldela** x ← right[x]
7. parent[s] ← y
8. **om** y=NIL **så tilldela** root[P] ← s
9. **annars om** key[s] < key[y] **så tilldela** left[y] ← s
10. **annars tilldela** right[y] ← s

Tilläggningsalgoritmen går vidare i trädet på samma sätt om sökalgoritmen. Algoritmen söker platsen där den nya noden borde läggas till. Ifall trädet är tomt, görs den nya noden till rot för trädet. Annars placeras denna på sin plats antingen som det vänstra eller högra barnet till någon nod som redan finns i trädet. För enkelhetens skull antar vi att man inte försöker lägga till ett nyckelvärde som redan finns i trädet.

Noden med det minsta värdet i det binära sökträdet hittas enligt följande:

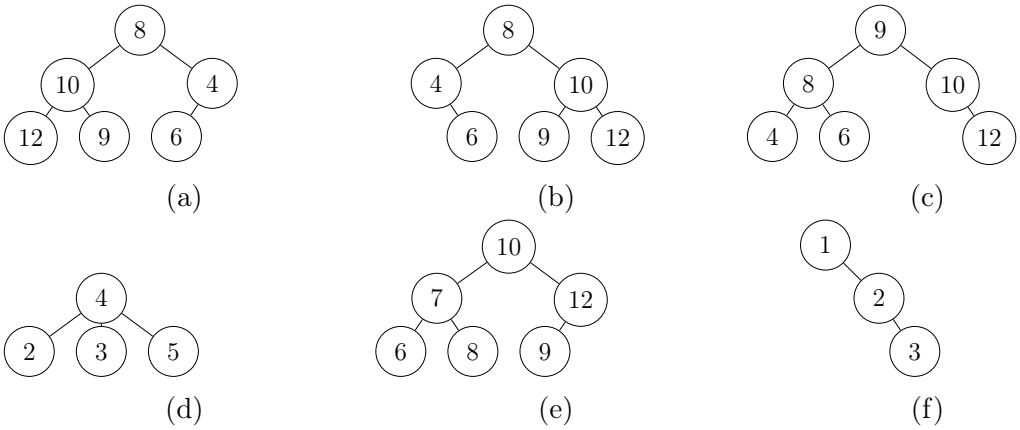
MinValue(träd P)

1. **tilldela** x ← root[P]
2. **om** x≠ NIL **så så länge som** left[x]≠ NIL **utför rad** 3
3. **tilldela** x ← left[x]
4. **resultatet är** x

Ifall trädet är tomt, finns det inget minsta värde och resultatet är NIL.

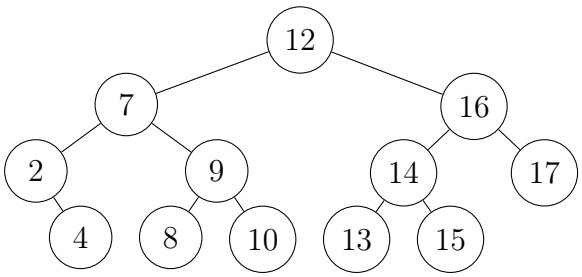
Fråga 1

Ge en kort motivering till vilka av strukturerna i bilderna (a-f) inte motsvarar binära sökträd (så som de är definierade i den här uppgiften). (maximalt antal poäng 4)



Fråga 2
På bilden finns ett binärt sökträd P. Beskriv för följande algoritmanrop vilka trädets noder som genomgås och i vilken ordning (svaret kan ges till exempel i formen "Anropet går igenom noderna i denna ordning: 15, 14, 16, 17"). (maximalt antal poäng 3)

a) MinValue(P)
b) SearchTree(P, 15)
c) SearchTree(P, 11)



Fråga 3
Rita det binära sökträd som uppstår då man lägger till följande noder i denna ordning till ett tomt träd: 5, 7, 12, 9, 8, 6, 16, 14, 20, 10, 1, 0 (talen är nyckelvärdena för noderna som läggs till). (maximalt antal poäng 5)

Fråga 4
Höjden på ett binärt sökträd är antalet steg man går kortaste vägen från roten till ett löv som ligger längst ned. Trädet i fråga 2 har höjden 3. Ett träd bestående av en enda nod har höjden 0. Höjden för ett tomt träd är inte definierad.

a) Hur många noder kan det lagras som mest i ett träd av höjden tio?
b) Bilda en sådan matematisk funktion f att värdet $f(h)$ berättar hur många noder som maximalt kan lagras i ett träd av höjden h. (Antag: $h \geq 0$)
c) Låt ett binärt sökträd med n noder vara uppbyggt så att dess höjd är så liten som möjligt. Hur många operationer (variabeltilldelningar, villkorskontroller osv.) måste algoritmen SearchTree utföra som mest i ett sådant träd (storleksordningen räcker).

(maximalt antal poäng 6)

Fråga 5
Om binärträdets noder räknas upp i storleksordning enligt sina nyckelvärden från mista till största, så är *följande nod* den nod som har följande nyckelvärde i ordningen. Till exempel, om noderna är $\{2, 4, 6, 7, 10\}$, så är följande nod till noden med nyckeln 4, noden med nyckeln 6. Noden med den största nyckeln har ingen följande nod. Skriv ut algoritmen Successor(nod s) som

- får som input en nod s som redan finns i trädet
- ger som resultat följande nod till noden s, eller värdet NIL om det inte finns någon följande nod

Kom ihåg att du når andra noder lagrade i trädet genom referenserna left[s], right[s] och parent[s] allt enligt behov. Innan du börjar skriva den detaljerade algoritmen, beskriv kort med ord hur din algoritm söker den följande noden och motivera varför din sökmetod fungerar rätt. Sträva i din lösning till effektivitet, dvs till att din algoritm går igenom endast den del av trädet som behövs (till exempel en algoritm som alltid går igenom hela trädet är inte effektiv). (maximalt antal poäng 7)