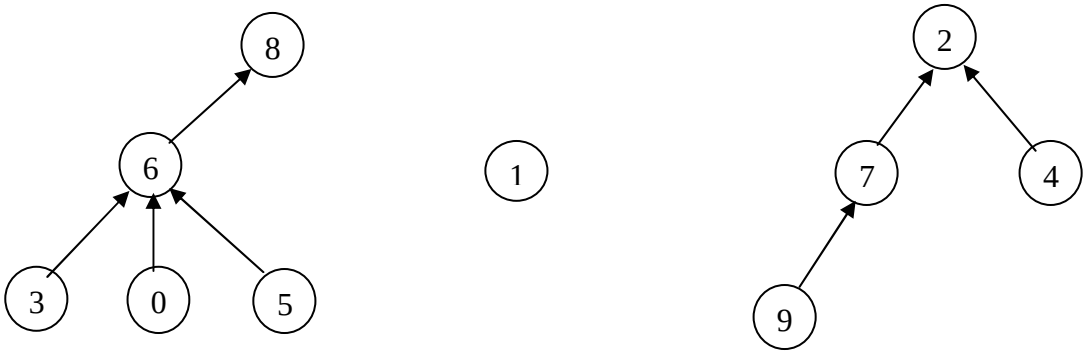
UNIVERSITETET I OSLO
Det matematisk-naturvitenskapelige fakultet

Eksamen i :	INF 110 — Algoritmer og datastrukturer
Eksamensdag :	Torsdag 5. desember 2002
Tid for eksamen :	09.00 - 15.00
Oppgavesettet er på :	6 sider
Tillatte hjelpemidler :	Alle trykte og skrevne

Kontroller at oppgavesettet er komplett før du begynner å besvare det

Oppgave 1 (40%)

Betrakt en mengde med n+1 noder. La nodene være identifisert ved tallene 0, 1, 2, … , n. Nodene er strukturert i en skog av rotrettede trær (hver node er med i ett tre). Se eksempel med n=9.



Oppgave 1.1

Beskriv hvordan nodestrukturen kan representeres ved hjelp av en heltallsarray:

```
int [] foreldre = new int[n+1];
```

Angi innholdet av `foreldre` for eksemplet ovenfor.

Oppgave 1.2

Programmer en metode

```
public void veiMotRot(int node)
```

som skriver ut veien fra noden gitt ved parameteren `node` opp til roten av det tilhørende treet.
I eksempelet kan `veiMotRot(3)` skrive ut:

```
Fra node:3 til node:6 til node:8 som er rot.
```

Oppgave 1.3

Programmer en metode

```
public void veiFraRot(int node)
```

som skriver ut veien fra roten av det tilhørende treet ned til noden gitt ved parameteren `node`.
I eksempelet kan `veiFraRot(3)` skrive ut:

```
Fra rot:8  til node:6  til node:3
```

Oppgave 1.4

Betrakt følgende selskapslek. På et bord står det ***n*** glass med stetten ned. Målet er å få snudd alle glassene på hodet ved å utføre en serie med snuoperasjoner. Bare en type snuoperasjon kan benyttes: snu samtidig ***s*** fritt valgte glass, hvor ***s*** er et fast tall (1 <= ***s*** <= ***n***).
For eksempel med ***n*** = 4 og ***s*** = 3:

Start: 4 opp, 0 ned. Snu 3 oppGlass og 0 nedGlass
gir: 1 opp, 3 ned. Snu 1 oppGlass og 2 nedGlass
gir: 2 opp, 2 ned. Snu 1 oppGlass og 2 nedGlass
gir: 3 opp, 1 ned. Snu 3 oppGlass og 0 nedGlass
gir: 0 opp, 4 ned.

Tilstanden under utførelse av selskapsleken kan enkelt representeres ved et heltall ***opp*** som angir antallet oppGlass i øyeblikket. I denne tilstanden har spilleren valget mellom flere snuoperasjoner.
Hva blir neste tilstand dersom antallet oppGlass som inngår i operasjonen er ***i***?
Hvilke restriksjoner er det på antallet oppGlass i operasjonen (uttrykt ved ***opp*** og ***s***)?

Oppgave 1.5

Skriv et fullstendig program med innparametre ***n*** og ***s*** som løser problemet. Programmet skal produsere en utskrift tilsvarende den i eksemplet ovenfor (eller en melding når det ikke finnes en løsning). Programmet avslutter når en løsning er funnet. (**HINT**: se oppgave 1.1).
For å få full uttelling på besvarelsen, må programmet finne en løsning med færrest mulige snuoperasjoner.

Oppgave 1.6

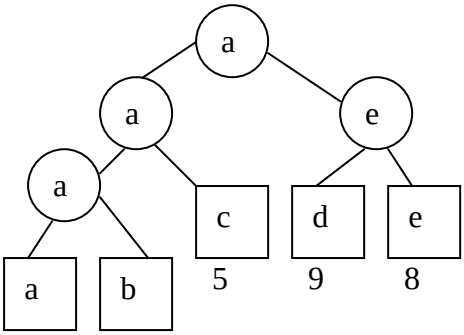
Anta at antall glass, ***n***, er et oddetall og antall glass i en snuoperasjon, ***s***, er et partall. Bevis at det da ikke finnes en løsning.

Oppgave 2 (40%)

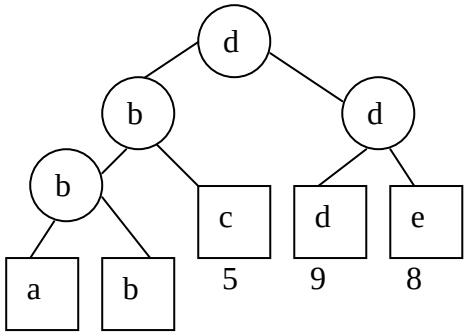
I denne oppgaven vil vi se på turneringstrær eller vinnertrær.
Definisjon: Et ***vinnertre*** for ***n*** spillere er et komplett binærtre med *n* blad og *n-1* indre noder. Hver indre (kamp) node identifiserer vinneren av kampen spilt mellom dens to barn.

Vi antar at hver spiller har en tilordnet verdi, og at det er en regel som avgjør vinneren basert på en sammenlikning av de to spilleres verdi. I et ***min-vinnertre*** er vinneren den spiller med den minste verdien, mens i et ***maks-vinnertre*** er vinneren den spiller med den største verdien. I tilfeller med lik verdi er vinneren, konsistent for hele turneringen, enten det venstre barnet eller det høyre barnet til kamp-noden.

Følgende figur viser maks- og min-vinnertrær for 5 spillere a, b, c, d og e med tilordnede verdier 4, 6, 5, 9 og 8, henholdsvis.



maks-vinnertre



min-vinnertre

En behagelig egenskap til et vinnertre er at det er lett å modifisere treet for å ivareta endring i en av spillerne. Endring i en spiller berører bare de kampene han/hun spiller. Etter endring av verdien til spilleren vil i beste fall ingen kampnoder endres og i verste fall må endringer skje i kampnodene fra spilleren til roten, i alt $\lceil \log_2 n \rceil$ endringer.
Abstrakt datatype for vinnertrær defineres som følgende:

AbstractDataType *Winner Tree* {
    instances
        komplett binærtrær hvor hver indre node peker til vinneren av kampen gitt av noden. Bladnoder representerer spillerne.
    operations
        initialize(a): initialiser et vinnertre for spillere i array a
        getWinner(m): returner vinneren av den indre noden m
        rePlay(i): spiller på nytt kamper etter en endring i spiller i
}

Oppgave 2.1
Tegn maks- og min-vinnertrær for spillere 2, 8, 3, 7, 20, 4, 3 and 9. Tegn så trærne som oppstår når 20 endres til 1. Få de nye trærne ved å spille på nytt bare de kampene på stien til roten.

Oppgave 2.2
Anta at et min-vinnertre er initialisert med n elementer (verdier) som spillere. min-vinnertrær kan bli brukt til å sortere disse n elementene (verdiene). Vær forsiktig å ikke ødelegge strukturen på treet, dvs ikke fjern noder slik som det for eksempel gjøres med heaps. Skriv pseudokode for en sorteringsalgoritme som bruker min-vinnertrær som datastruktur.

Oppgave 2.3
Hva er øvre og nedre grense for kjøretiden av din algoritme?

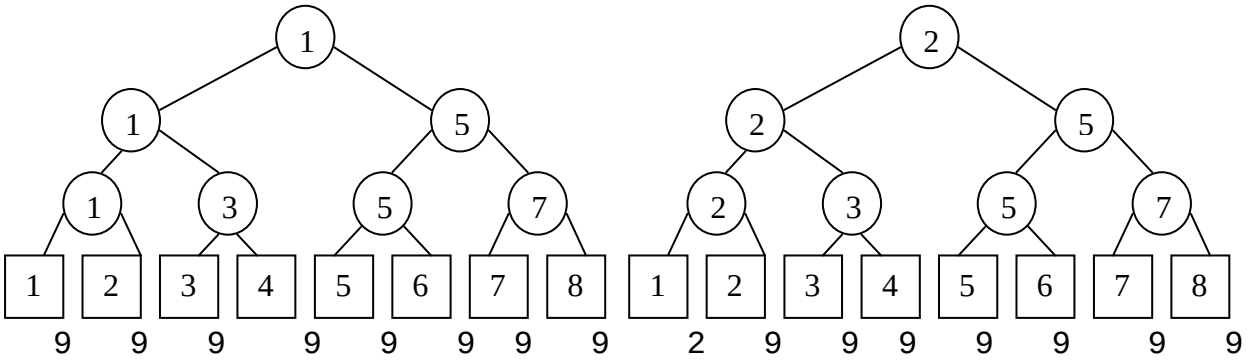
Oppgave 2.4
Vi betrakter nå en annen applikasjon av vinnertrær. De blir brukt for å gi en approksimasjonsalgoritme for det berømte NP-problemet: ”bin-packing”. Problemet er beskrevet i læreboka, kapittel 10, men for kompletthet såvel for noen forenklinger, gis en fullstendig beskrivelse av problemet her.

I ”bin-packing”-problemet har vi et ubegrenset tilgang til ”bins” av en fast kapasitet binCapacity og ***n*** objekter som skal plasseres i disse ”bins”. Objekt i krever objectSize[i] enheter av kapasitet. Forutsetning: 0 ≤ objectSize[i] ≤ binCapacity (merk at n ”bins” er tilstrekkelig for å pakke n objekter av størrelse mindre enn binCapacity, men poenget er å benytte så få som mulig). En **aksepterbar** ”bin-packing” er en tilordning av objekter til ”bins” slik at kapasiteten til ingen ”bins” overskrides. En **aksepterbar** ”bin-packing” som benytter færest antall ”bins” er en **optimal** ”bin-packing”.

”First Fit”-algoritmen er en av approksimasjonsalgoritmene som løser problemet, men som ikke garanterer en optimal løsning.
”First Fit” virker som følgende: anta at ”bins” arrangeres fra venstre til høyre og er nummerert fra *1* til *n* (merk: her er 1 til n **navn** på ”bins”, ikke verdier som brukes til sammenlikning). Objekt i pakkes i første ”bin”, testet fra venstre, hvor det er plass.

Dette kan implementeres ved hjelp av maks-vinnertrær. Først initialiseres et maks-vinnertre med *n* spillere. Spiller i har tilordnet som verdi resterende kapasitet til ”bin” i. Denne kapasiteten er initielt lik binCapacity for alle ”bins”. Metoden antar at når en kamp avgjøres, venstre spiller vinner unntatt når høyre spiller har høyere verdi. Objektene skal tilordnes ”bins” en etter en. Når objekt i blir tilordnet, følges en sti fra rota til den ”bin” lengst til venstre (lavest nummer) som har plass til objektet. Figuren nedenfor viser maks-vinnertreet med 8 ”bins”, initiert med binCapacity lik 9 og et tre etter at første objekt av størrelse 7 er tilordnet.

Tegn treet etter tilordning av de resterende objekter av størrelse 2, 5, 3, 6, 6, 1 og 8.



Oppgave 2.5

Inspirert av hva du har lært av eksemplet ovenfor, skal en ”First Fit”-algoritme skrives i Java. Anta følgende:

```
public class Player {
  int name;
  int value;
  public Player(int name, int value) {
    this.name=name;
    this.value=value;
  }
}
```

```
public interface WinnerTree {

  // Builds a max winner tree with players given by
  // the parameter thePlayers.
  // Assume that the complete binary tree is represented
  // as an array of Player objects.
  public void initialize(Player[] thePlayers);

  // int i is the index into the complete binary tree.
  // If i identifies a match node, returns the winner of the
  // match.
  // If i identifies one of thePlayers, returns the given
  // player.
  public Player getWinner(int i);

  // int i is the index into the complete binary tree
  // identifying one of thePlayers.
  // Replays the matches starting from the given player.
  public void rePlay(int i);
}
```

Basert på koden ovenfor, skal det skrives en metode `firstFitPack` for å pakke objekter i henhold til en ”First Fit”-strategi.

```
public static void firstFitPack(int[] objectSize,
       int binCapacity, WinnerTree maxWT) { . . . }
```

hvor:

`objectSize`	størrelsen på objektene som skal pakkes.
`binCapacity`	kapasiteten til ”bins”.
`maxWT`	et objekt av en klasse som implementerer interface WinnerTree.

Oppgave 3 (20%)

I denne delen er det ikke krav å skrive noe Java kode. Du kan besvare ved hjelp av pseudokode eller i form av korte beskrivelser.

Oppgave 3.1

Betrakt en **urettet** graf G. Beskriv en algoritme som tester om G er asyklisk eller ikke. Hvis ikke, skal algoritmen finne det minste antallet kanter som må fjernes fra G for å få en asyklisk graf.

Oppgave 3.2

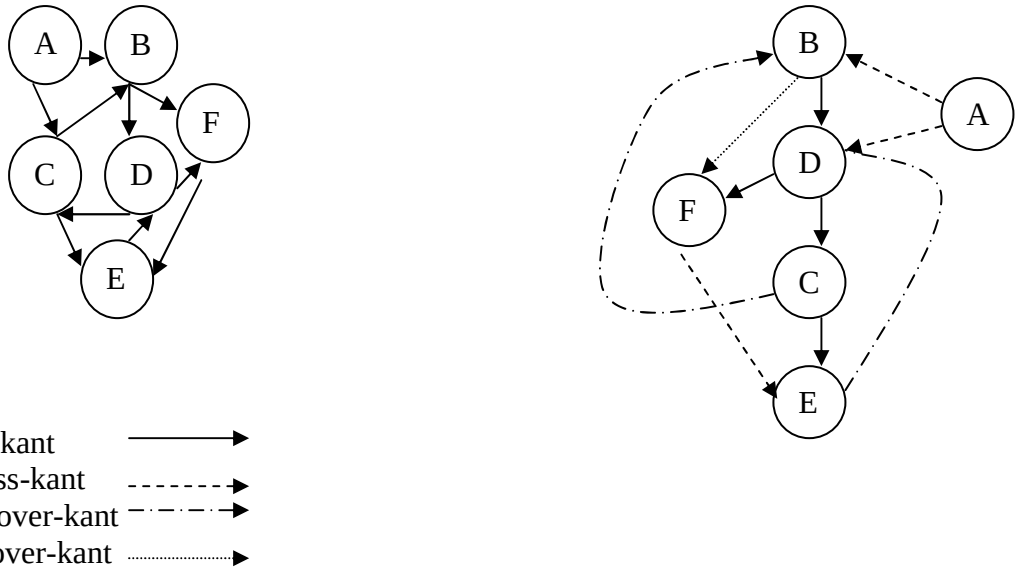
Betrakt en **rettet** graf G. I løpet av et dybde-først søk i G med start i et gitt node s, kan hver kant som behandles bli klassifisert som en av de følgende fire typer (i relasjon til den spann skogen (spanning forest) som blir konstruert av dybde-først søket):

1) tre-kant (som leder fra den nåværende noden til en ny umerket node. Tre-kantene vil etter søket representere en spann skog med s som rot i det første treet)
2) bakover-kant (fra den nåværende noden til en merket node m høyere opp i treet. Dvs det et en sti av tre-kanter som leder fra m til nåværende node)
3) forover-kant (fra den nåværende noden til en merket node m lavere ned i treet. Dvs det et en sti av tre-kanter fra nåværende node til m)

4) kryss-kant (fra den nåværende noden til en merket node m slik at det ikke er noen sti (av tre-kanter) fra nåværende node til m eller fra m til den nåværende noden).

Beskriv en algoritme som avgjør, gitt G og en startnode s, for enhver kant (u,v) i G om (u,v) er en tre-kant, bakover-kant, forover-kant eller en kryss-kant.

Tips: du kan tilordne nummer til nodene i spann skogen i løpet av dybde-først søket.



Eksempel: Rettet graf G og den resulterende dybde-først skogen startende i node B.

Oppgave 3.3

Beskriv en algoritme som finner det minste antallet av kanter som må fjernes fra en rettet graf G slik at den resulterende grafen er asyklisk.