ALGORITMIEK: opgaven werkcollege 2

BINAIRE BOMEN

Er moeten enige **recursieve** functies geschreven worden die werken op binaire bomen. We bekijken binaire bomen die behalve een info-veld van type char ook een extra niveau-veld van type integer bevatten. Een knoop van zo'n binaire boom ziet er dus als volgt uit:

```
class knoop {
  knoop* links,  // Linker kind.
  knoop* rechts; // Rechter kind.
  char info;
  int niveau;    // Niveau van een knoop in de boom
}; //knoop
```

Er is al een skeletprogramma beschikbaar, met een klasse binaireboom, waarvan nog niet alle functies zijn ingevuld. Zie: `https://liacs.leidenuniv.nl/~vlietrvan1/algoritmiek/` Van de binaire bomen in de testvoorbeelden zijn de niveau-velden nog niet geïnitialiseerd.

Opgave 1. Schrijf een recursieve memberfunctie `int aantalbladeren_p(knoop* ingang)`, die het aantal bladeren bepaalt van de binaire (sub-)boom met wortel `ingang`.

Opgave 2. Schrijf een recursieve memberfunctie `int hoogte_p(knoop* ingang)` die de hoogte berekent van de binaire (sub-)boom met wortel `ingang`. De hoogte van een binaire boom is het grootste niveau dat voorkomt, waarbij de wortel van de boom op niveau 0 zit.

Opgave 3. Schrijf een recursieve memberfunctie `void initniveau_p(knoop* ingang)`, die de niveau-velden van de binaire (sub-)boom met wortel `ingang` allemaal op 0 zet.

Opgave 4. Schrijf een recursieve memberfunctie `void vulniveau_p(knoop* ingang, int niveau)`, die het niveau-veld in elke knoop van de binaire (sub-)boom met wortel `ingang` vult met het niveau van die knoop. Je mag aannemen dat de parameter `niveau` bij de eerste aanroep van de functie het correcte niveau is voor de knoop `ingang`.

Opgave 5. Schrijf een recursieve memberfunctie `char maxinfowaarde_p(knoop* ingang)` die de maximale info-waarde (dat is hier het karakter met de hoogste ASCII-waarde) bepaalt die in de binaire (sub-)boom met wortel `ingang` zit.

Opgave 6. Schrijf een recursieve memberfunctie `void doepostorde_p(knoop* ingang)`, die de inhoud van de binaire (sub-)boom met wortel `ingang` in *postorde*-volgorde afdrukt. Van alle knopen wordt (in postorde-volgorde dus) het info-veld en het niveau-veld afgedrukt, als volgt:
info, niveau
info, niveau
info, niveau
...
Je mag aannemen dat deze functie pas wordt aangeroepen als de niveau-velden gevuld zijn.

BINAIRE ZOEKBOMEN

We gaan naar een toepassing van binaire bomen kijken, namelijk de *binaire zoekboom*. Deze bomen hebben de eigenschap dat in elke knoop geldt dat alle knopen in de linkersubboom een kleinere waarde bevatten dan de knoop zelf, en alle knopen in de rechtersubboom een grotere waarde dan de knoop zelf. Afgezien van opgave 10 moeten de functies die je schrijft **niet** recursief zijn, en mag je aannemen dat het om een binaire zoekboom gaat.

Opgave 7. Schrijf een memberfunctie `knoop* bzoek_p(char waarde, knoop* & ouder)`, die in de (hele) binaire zoekboom zoekt naar een knoop met inhoud `waarde` en de pointer naar die knoop retourneert. Na afloop van deze functie moet `ouder` wijzen naar de ouder van de gevonden knoop.
Als de waarde niet aanwezig is, moet `nullptr` geretourneerd worden (`ouder` wijst dan naar de knoop waar het zoeken ophoudt omdat de waarde niet aanwezig blijkt te zijn). Als de boom leeg is of als `waarde` in de wortel zit, moet `ouder` op `nullptr` blijven staan. Je mag aannemen dat de waarde van `ouder` bij aanroep `nullptr` is.

Opgave 8. Schrijf een memberfunctie `knoop* grootstekleinere(knoop* ingang, knoop* & ouder)`, die in de subboom met wortel `ingang` op zoek gaat naar de knoop met de grootste info-waarde kleiner dan de info-waarde van de knoop waar `ingang` naar wijst. De pointer naar deze knoop wordt geretourneerd. Na afloop van deze functie moet `ouder` wijzen naar de ouder van deze grootste kleinere. Als de grootste kleinere niet bestaat (binnen de boom met wortel `ingang`) moet `nullptr` geretourneerd worden. In dat geval wordt/blijft `ouder` `nullptr`.
Je mag aannemen dat de pointer `ingang` bij aanroep niet `nullptr` is, en dat de waarde van `ouder` bij aanroep juist wel `nullptr` is.
Hint: kijk goed naar de structuur van een binaire zoekboom om te achterhalen waar de grootste kleinere zich ten opzichte van de gegeven knoop `ingang` bevindt.

Opgave 9. Schrijf een memberfunctie `knoop* kleinstegrotere(knoop* ingang, knoop* & ouder)`, analoog aan de voorgaande opgave, die in de subboom met wortel `ingang` op zoek gaat naar de knoop met de kleinste info-waarde groter dan de info-waarde van `ingang` zelf.

Opgave 10. Schrijf een recursieve memberfunctie `bool isbzboom_p(knoop* ingang)`, die `true` teruggeeft als de binaire (sub-)boom met wortel `ingang` een binaire zoekboom is en `false` als dit niet zo is.
Hint: Gebruik ook de functies `grootstekleinere()` en `kleinstegrotere()`.

Opgave 11. Schrijf een memberfunctie `void bzvoegtoe_p(char waarde)`, die een waarde aan een binaire zoekboom toevoegt indien die nog niet aanwezig is. Let ook op het niveau-veld van de nieuwe knoop.
Hint: Gebruik de functie `bzoek_p()`. Gebruik verder de reeds in het skeletprogramma geïmplementeerde constructor `knoop(char waarde, int niv)`.

Opgave 12. Schrijf een memberfunctie `void bzverwijder_p(char waarde)`, die een waarde uit een binaire zoekboom verwijdert, indien aanwezig.
Hint 1: Gebruik de functies `bzoek_p()` en `grootstekleinere()` of `kleinstegrotere()`.
Hint 2: Onderscheid de gevallen: blad, knoop met 1 kind en knoop met 2 kinderen.