Tweede college algoritmiek

12 februari 2016

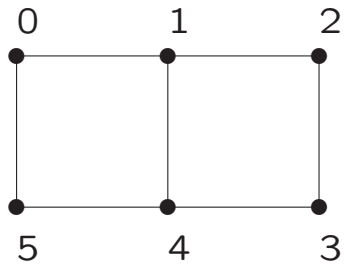
Grafen en bomen

Een graaf G wordt gedefinieerd als een paar (V, E), waarbij V een eindige verzameling is van knopen (vertices) en E een verzameling van paren van knopen: de takken/pijlen (edges). Een tak/pijl (u, v) verbindt de knopen u en v met elkaar. We bekijken meestal grafen zonder lussen.

Terminologie:

- ongerichte/gerichte graaf
- gewogen graaf
- paden en cykels/kringen
- samenhangend
- acyclisch

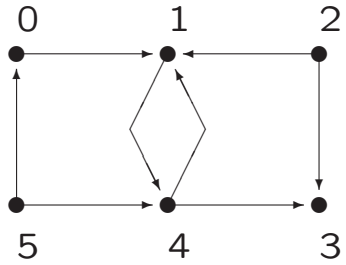
Grafen: voorbeelden



1.

$V = \{0, 1, 2, 3, 4, 5\}$;

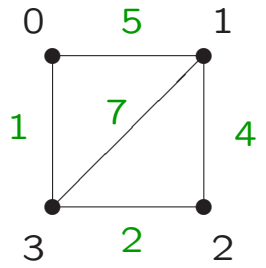
$E = \{(0, 1), (1, 2), (2, 3), (3, 4), (4, 5), (5, 0), (4, 1)\}$



2.

$V = \{0, 1, 2, 3, 4, 5\}$;

$E = \{(0, 1), (2, 1), (2, 3), (4, 3), (5, 4), (5, 0), (4, 1), (1, 4)\}$



3.

$V = \{0, 1, 2, 3\}$;

$E = \{(0, 1), (1, 2), (2, 3), (0, 3), (1, 3)\}$

Adjacency matrix: de **gerichte** graaf wordt gerepresenteerd door een tweedimensionaal array `int graaf[n][n]` (n het aantal knopen), waarbij `graaf`$[i][j]$ aangeeft of er een pijl van i naar j gaat. (Analoog ongerichte grafen; zie college 1.)

Adjacency list: de **gerichte** graaf wordt gerepresenteerd door een eendimensionaal array `buur* graaf[n]` (n het aantal knopen), waarbij `graaf`$[i]$ de ingang is tot een lijst van knopen waarvoor er een pijl is van i naar die knoop. De buurlijst bevat dus alle uitgaande pijlen uit i. (Analoog ongerichte grafen; zie college 1.)

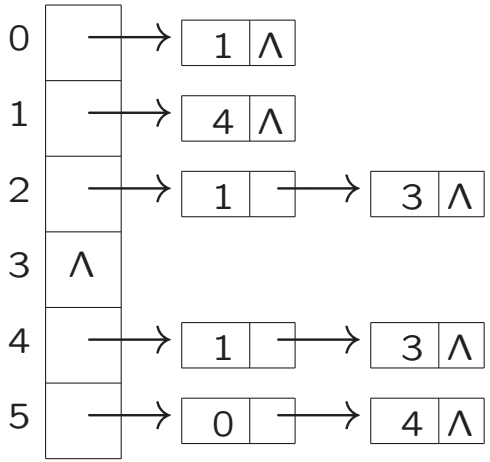
Adjacency list representatie in C++:

```
class buur { public:
  int knoopnummer;
  buur* volgende;
}  // misschien meer velden, bv. char info of int gewicht
buur* graaf[n];
```

Adjacency matrix en adjacency list voor voorbeeldgraaf **2**:

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

adjacency matrix



adjacency list

Geef een algoritme dat de pijl (i, j) in een gegeven gerichte graaf omdraait. Neem aan dat de omgekeerde pijl (j, i) nog niet bestaat.

```
// het array graaf (buur* graaf[n]) bevat de gerichte graaf
buur* hulp = graaf[i]; buur* vorige = NULL;
while ( hulp->knoopnummer != j ) { // knoop j zoeken
    vorige = hulp; hulp = hulp->volgende;
}
// haal buurknoop j uit lijst
if ( vorige == NULL ) { // j is de eerste buur van i
   graaf[i] = hulp->volgende;
}
else { // j is niet de eerste buur van i
   vorige->volgende = hulp->volgende;
}
delete hulp;  // gooi buur j weg
// voeg nu i vooraan in de buurlijst van j toe
hulp = new buur;
hulp->knoopnummer = i; hulp->volgende = graaf[j];
graaf[j] = hulp;
```

Gegeven een adjacency list representatie van een gerichte graaf: hoeveel stappen kost het om het aantal uitgaande pijlen van iedere knoop te vinden? En het aantal inkomende pijlen?

En nu dezelfde vraag, maar voor een adjacency matrix representatie?

Hoeveel stappen kost het om te testen of van de ene knoop een pijl loopt naar de andere bij een adjacency list respectievelijk een adjacency matrix representatie?

Hoeveel ruimte neemt een adjacency list in beslag? En een matrix?

- Een pad van u naar v is een rij knopen waarvoor geldt dat tussen elk tweetal opeenvolgende knopen uit die rij een tak zit (resp. een pijl loopt van de ene naar de volgende knoop; dan: gericht pad).

- De lengte van een pad = het aantal takken op dat pad = het aantal knopen - 1.

- Als alle knopen verschillend zijn heet het pad **enkelvoudig**.

- Een kring in een ongerichte graaf is een pad dat begint en eindigt in dezelfde knoop en dat geen enkele tak meer dan één keer bevat.

- Een ongerichte graaf heet samenhangend als er tussen elk tweetal knopen u en v een pad loopt.

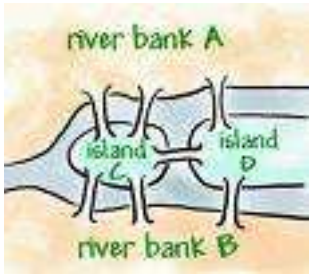
- Een graaf die geen kringen bevat heet acyclisch.

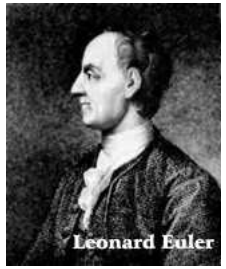
- Een graaf die samenhangend en acyclisch is noemen we een boom

- In een boom is er precies één pad tussen elk tweetal knopen u en v.

Twee zeer bekende graafproblemen:

Koningsberger bruggen probleem





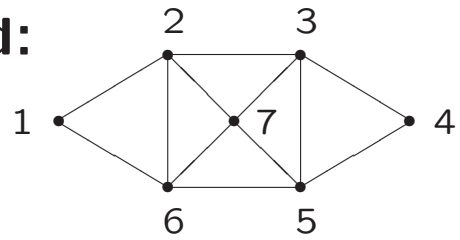
Vierkleurenprobleem

Hoezo **graaf**problemen?

Definitie: een wandeling (pad) in een ongerichte graaf die terugkeert in zijn beginpunt en die *alle* takken precies één keer doorloopt heet een Eulerkring*. Analoog: Eulerpad.

Probleem. Gegeven een ongerichte graaf $\mathcal{G}$. Heeft $\mathcal{G}$ een Eulerkring?

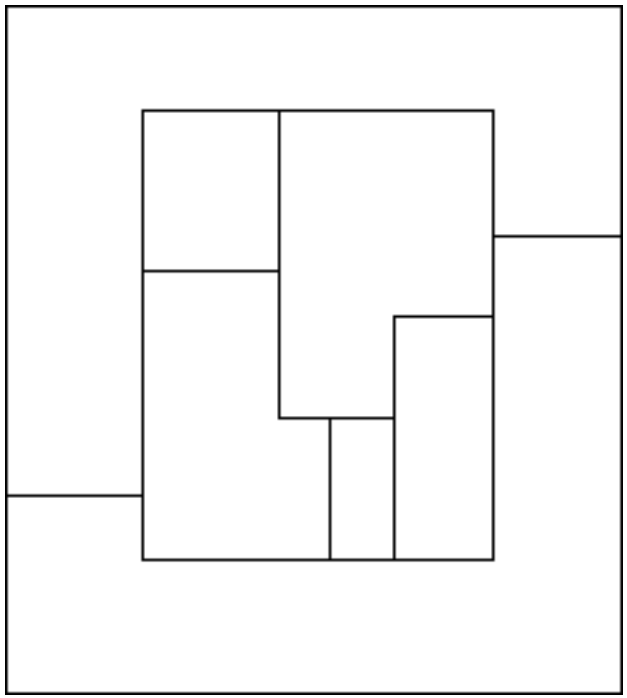
Voorbeeld:



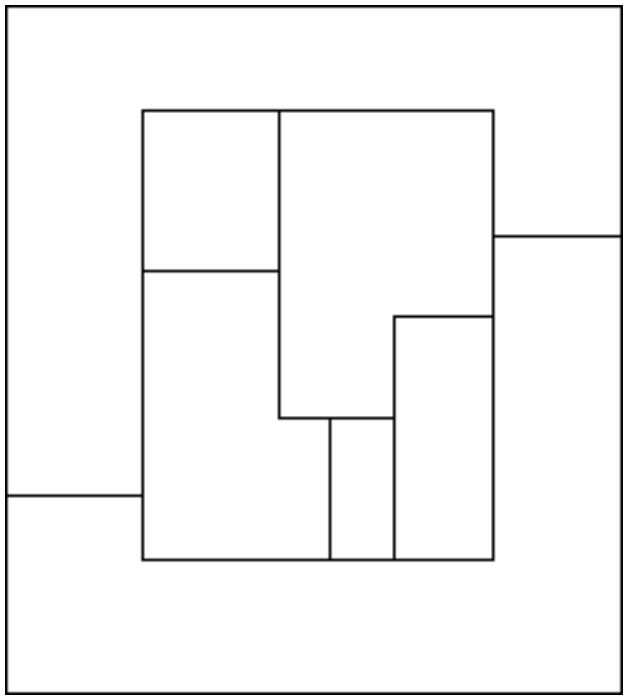
Voor deze graaf is 1 2 3 4 5 3 7 5 6 7 2 6 1 een Eulerkring.

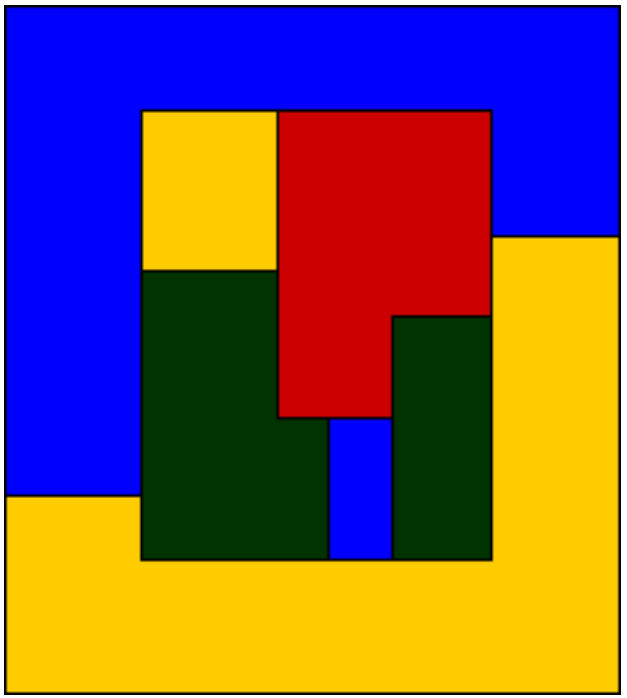
* Een kring is een pad dat begint en eindigt in dezelfde knoop en dat geen enkele tak meer dan één keer doorloopt.

Kleur de landkaart met maximaal vier kleuren onder de restrictie dat buurlanden een verschillende kleur hebben:



Kleuring van de landkaart met maximaal vier kleuren onder de restrictie dat buurlanden een verschillende kleur hebben:

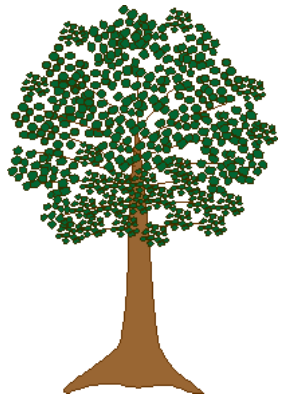




Definitie: een boom is een samenhangende (= uit één stuk bestaande) ongerichte graaf zonder cykels (= kringen).

Wijs een speciale knoop aan, de *wortel*. Teken de wortel bovenaan en alle paden vanuit de wortel naar beneden: dit geeft een hiërarchische structuur die lijkt op een stamboom. Dit heet ook wel een georiënteerde boom. Meestal spreken we gewoon van een boom.

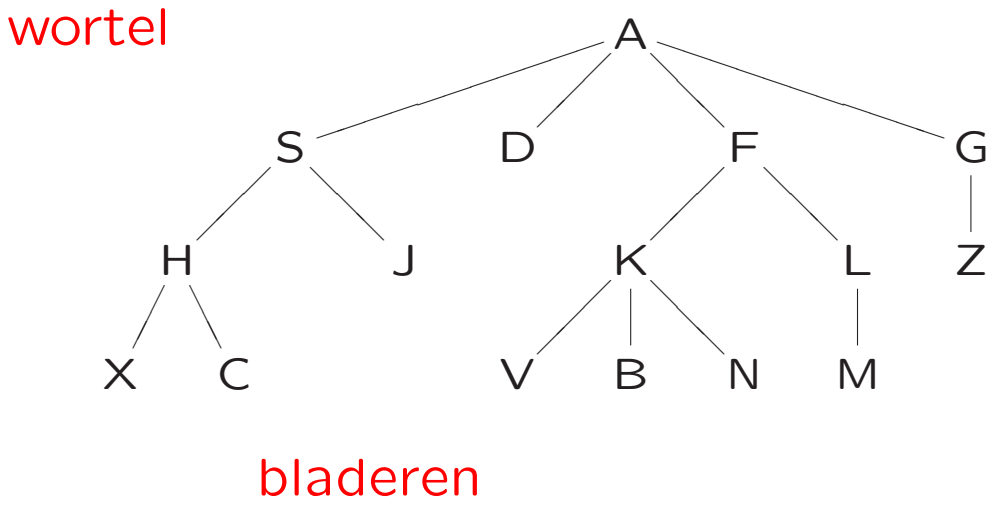
Stamboomterminologie: kind $\longleftrightarrow$ ouder, afstammeling $\longleftrightarrow$ voorouder.



In een georiënteerde boom hebben we dus ouder-kind relaties tussen knopen.

Voorbeeldboom

Terminologie: takken, knopen, niveau, hoogte, wortel, bladeren $\longleftrightarrow$ interne knopen

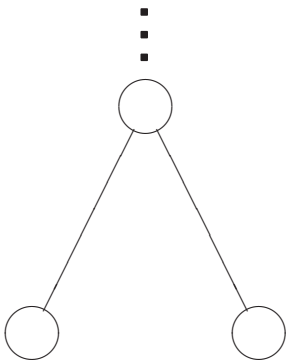


De wortel (hier A) is de *enige* ingang tot de boom.

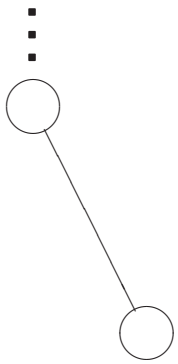
De wortel bevindt zich op niveau 0 (afspraak). De hoogte van de boom is het grootste niveau dat voorkomt, ofwel de lengte van het langste pad van de wortel tot een blad; dat is hier dus 3.

- Een acyclische graaf die niet samenhangend is heet een bos. Het is namelijk een collectie bomen.
- Voor bomen geldt: aantal takken = aantal knopen - 1.
- Een geordende boom is een boom waarin van elke knoop de kinderen geordend zijn: oudste kind, een na oudste kind, ..., jongste kind.
- Onderstaande bomen zijn als georiënteerde bomen wel gelijk, maar als geordende georiënteerde bomen niet:

Een binaire boom is een boom waarin elke knoop ofwel nul, ofwel één ofwel twee kinderen heeft; als een knoop twee kinderen heeft dan is het ene kind het **linkerkind**, het andere het **rechterkind**; als een knoop één kind heeft, dan is dit ofwel een linkerkind, ofwel een rechterkind.

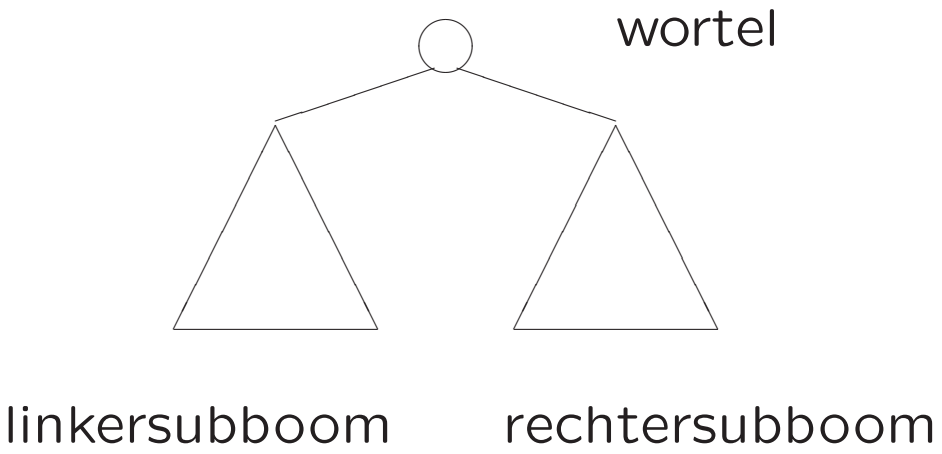


linkerkind rechterkind



één kind: rechterkind

Recursieve definitie: een binaire boom is een eindige verzameling knopen die ofwel leeg is, ofwel bestaat uit een speciale knoop (de wortel) en twee disjuncte verzamelingen knopen die samen de rest van alle knopen vormen. Die knoopverzamelingen vormen beide ook weer een binaire boom: de linkersubboom en de rechtersubboom.



```
class knoop {
  public:
    knoop ( ) {  // constructor
      info = 0; links = NULL; rechts = NULL; }
    int info;
    knoop* links;
    knoop* rechts;
};  // knoop
```

De binaire boom wordt gerepresenteerd door middel van een pointer naar de wortel:

```
knoop* wortel; // de ingang tot de binaire boom
```

Netter om de binaire boom met een klasse te representeren: zie Programmeermethoden en/of Bomenpracticum Algoritmiek.

WLR (preorde):

bezoek wortel
doorloop linkersubboom WLR
doorloop rechtersubboom WLR

LWR (symmetrisch):

doorloop linkersubboom LWR
bezoek wortel
doorloop rechtersubboom LWR

LRW (postorde): analoog

Algemene gedaante van een recursieve functie:

if basisgeval **then**

los op zonder recursie; // makkelijk

else

één of meer recursieve eenvoudigere aanroepen;

fi

```
void preorde (knoop* root) {
  if ( root != NULL ) {
    cout << root->info << endl;
    preorde (root->links);
    preorde (root->rechts);
  }  // if
}  // preorde

void symmetrisch (knoop* root) {
  if ( root != NULL ) {
    symmetrisch (root->links);
    cout << root->info << endl;
    symmetrisch (root->rechts);
  }  // if
}  // symmetrisch
```

We tellen *recursief* het aantal knopen van een binaire boom met ingang wortel.
Aanroep: `int tellen = aantal (wortel);`

```
int aantal (knoop* root) {
  if ( root == NULL )       // lege boom
    return 0;
  else
    return ( 1 + aantal (root->links)
               + aantal (root->rechts) );
}  // aantal
```

Merk op dat hier eigenlijk een preorde-wandeling wordt gedaan.

We breken de binaire boom met ingang wortel helemaal af: hiertoe wordt eerst de linkersubboom (recursief) weggegooid, daarna de rechtersubboom en ten slotte de wortel zelf. Aanroep: `breekaf (wortel);`

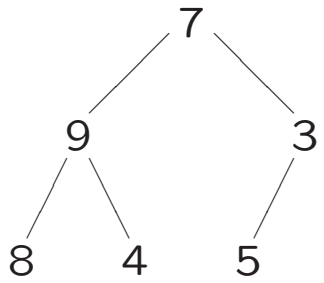
```
void breekaf (knoop* & root) {
  if ( root != NULL ) {
    breekaf (root->links);      L
    breekaf (root->rechts);     R
    delete root;                W
    root = NULL;
  }  // if
} // breekaf
```

- Voor de hoogte h van een binaire boom met n knopen geldt: $\lfloor \log_2 n \rfloor = \lceil \log_2(n+1) \rceil - 1 \leq h \leq n - 1$

- Een complete binaire boom is een binaire boom waarbij alle niveaus geheel vol zitten, behalve eventueel het onderste. Op het onderste niveau mogen alleen de meest rechter knopen missen.

- Een binaire zoekboom is een binaire boom waarbij voor elke knoop geldt dat de waarde in die knoop groter is dan alle waarden in zijn linkersubboom, en kleiner dan alle waarden in zijn rechtersubboom.

Complete binaire boom:

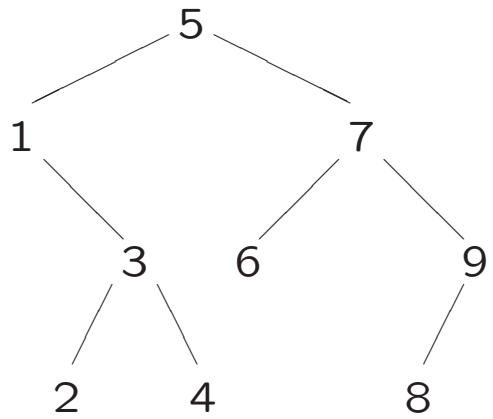


$\Longleftrightarrow$ 7 9 3 8 4 5

Binaire zoekboom:

LWR

1 2 3 4 5 6 7 8 9



- **Lezen/leren uit Levitin bij dit college:**

  Paragraaf 1.4 (vanaf graafrepresentaties)

- **Werkcollege:** Bomenpracticum

  donderdag 18 februari 2016, 13:45—15:30, computer-zalen 306/308, 302/304

- **Opgaven:**

  zie liacs.leidenuniv.nl/~graafjmde/ALGO/

- Controleer de website voor de eerste programmeeropdracht

- **Volgend college:**

  vrijdag 19 februari 2016, 11:15—13:00, zaal B2