Eerste college algoritmiek

5 februari 2016

Introductie

- docent: Jeannette de Graaf; kamer 151
  e-mail: j.m.de.graaf@liacs.leidenuniv.nl

- assistenten: Hanjo Boekhout, Leon Helwerda, Ruud Heesterbeek, Bart van Strien

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

- college: vrijdag 11:15 – 13:00 in zaal B2

- werkcollege: donderdag 13:45 – 15:30 in zaal B1 en B2 (of computerzaal 306/308, 302/304)

- indeling van de werkgroepen wordt nog bekendgemaakt

- eerste college: vrijdag 5 februari 2016
- eerste werkcollege: donderdag 11 februari 2016
- tentamen: dinsdag 7 juni 2016, 14:00–17:00 uur
- hertentamen: donderdag 7 juli 2016, 10:00–13:00 uur

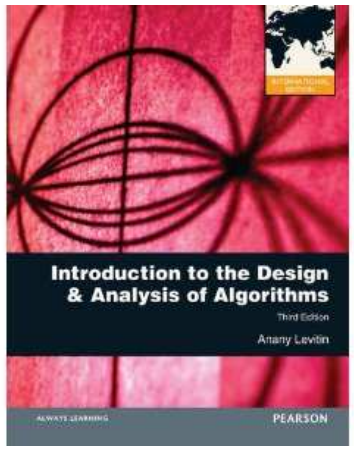
Bij dit college behorende boek:

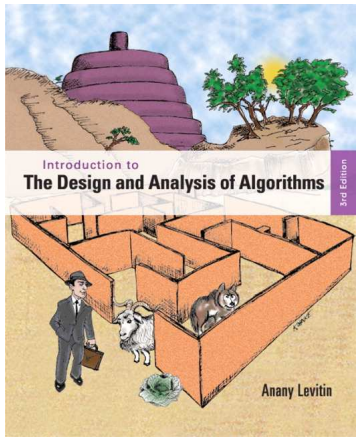
Anany Levitin

Introduction to The Design and Analysis of Algorithms

third edition (verwijslijst second edition op website)

Addison-Wesley, 2012





- drie programmeeropdrachten in C++
- nadruk op toepassen van besproken probleemoplossings-methoden
- bij voorkeur in tweetallen
- alle drie voldoende (≥ 6)
- ```
  if (tentamen gehaald)
    if (programmeerwerk in orde)
      eindcijfer = 2/3 tentamencijfer + 1/3 practicumcijfer;
    else
      (nog) geen eindcijfer;
  else
    eindcijfer = tentamencijfer;
  ```
  ```

Een algoritme is een opeenvolging van **ondubbelzinnig** omschreven instructies die leiden tot een oplossing van een probleem. Het algoritme geeft voor elke willekeurige legale invoer van het probleem de vereiste uitvoer in een **eindig** aantal stappen (in eindige tijd).

In de praktijk willen we dat een algoritme **effectief** is, dat wil zeggen binnen redelijke tijd klaar is, en liefst zo snel mogelijk (**efficiëntie**): complexiteit van een algoritme.

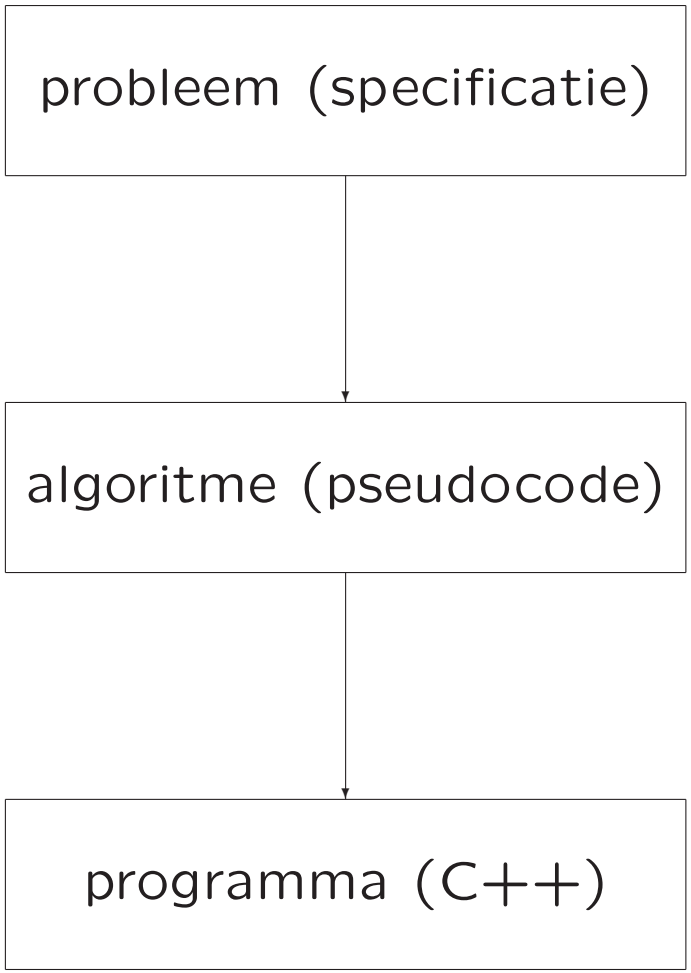
Het woord algoritme is afgeleid van Abu Ja'far Muhammed ibn Musa al-Khwarizmi (ca. 780–850), een belangrijk Arabisch wiskundige uit de middeleeuwen.


ПОЧТА СССР
4к
1983
1200
ЛЕТ
Мухаммед
аль·Хорезми

Wat is een algoritme?

Een algoritme is een recept of methode voor het oplossen van een probleem, met de volgende eigenschappen:

1. **eindigheid**: stopt na een eindig aantal stappen
2. **bepaaldheid**: bestaat uit ondubbelzinnig omschreven stappen
3. **invoer**: het is duidelijk wat geldige invoer is
4. **uitvoer**: er kan bewezen worden dat de correcte uitvoer wordt gegeven voor elke geldige invoer
5. **effectiviteit**: de stappen zijn voldoende eenvoudig en het antwoord wordt hopelijk binnen redelijke tijd gegeven



ontwerp

implementatie

Algoritmiek en verder

- ontwerpen van algoritmen: Algoritmiek, Datastructuren
- hoe beschrijven we het algoritme?
- correctheid bewijzen: Programmeren & correctheid
- analyse van algoritmen: Complexiteit
  - efficiëntie
    . theoretische analyse: tijdcomplexiteit, ruimtecomplexiteit
    . empirische analyse: testen
  - optimaliteit: bestaat er een beter algoritme?

In dit college bekijken we enkele belangrijke ontwerptechnieken voor het algoritmisch oplossen van problemen:

* brute force/exhaustive search
* divide and conquer
* backtracking
* branch and bound
* dynamic programming
* greedy approach

Probleem: *Gegeven* twee niet-negatieve gehele getallen m en n (niet beide nul). *Vraag*: wat is de grootste gemeenschappelijke deler, genoteerd als ggd(m,n), van m en n?

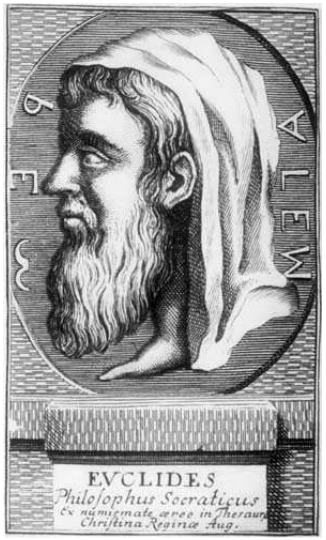
Voorbeeld: ggd(60,24) = 12; ggd(25,0) = 25;
ggd(200, 441) = 1; ggd(495,588) = 3.

Het algoritme van Euclides is gebaseerd op het herhaald gebruiken van de gelijkheid

$$\text{ggd}(m, n) = \text{ggd}(n, m \bmod n),$$

totdat de tweede parameter nul wordt.

Voorbeeld: ggd(60,24) = ggd(24, 12) = ggd(12, 0) = 12



Euclides van Alexandrië (ca. 325 – 265 (voor Christus))

Prominent wiskundige uit de oudheid, vooral bekend vanwege zijn verhandeling over meetkunde: De Elementen.

Het algoritme van Euclides in pseudocode (anders dan die in het boek wordt gebruikt):

while $n \neq 0$ **do**
// elementaire operaties: deling en toekenning
    rest := m **mod** n;
    $m := n$;
    $n :=$ rest;
od
return m;

Zie ook dictaat en college Programmeermethoden.
En verder: Levitin, 1.1 opgave 6 t/m 9.

Er zijn nog andere algoritmen voor het berekenen van de ggd van twee getallen. We bekijken er twee. Welke van de drie is het efficiëntst?

- algoritme van Euclides
- iteratief controleren
- ontbinden in priemfactoren

Het volgende algoritme (genoteerd in gewone taal) berekent ggd(m,n) door van alle integers $\leq$ min(m,n) te proberen of ze m en n beide delen.

Algoritme 2:

Stap 1 Bepaal min(m,n) en ken de waarde toe aan t;
Stap 2 Deel m door t. Als de rest 0 is, ga naar Stap 3; anders ga naar Stap 4;
Stap 3 Deel n door t. Als de rest 0 is, dan is t de ggd, dus return t en stop; anders ga naar Stap 4;
Stap 4 Laag t met 1 af en ga naar Stap 2;

Merk op dat –in tegenstelling tot het algoritme van Euclides– dit algoritme fout gaat als m of n nul is. Er moet hier dus gelden: $m \neq 0$ *en* $n \neq 0$.

Onderstaande methode vindt de ggd door m en n te ontbinden in hun priemfactoren. De ggd is dan het product van de gezamenlijke priemfactoren.

Voorbeeld: $60 = 2^2 * 3 * 5$ en $24 = 2^3 * 3$, dus ggd$(60, 24)$ $= 2^2 * 3 = 12$.

Stap 1 Vind de priemontbinding van m;
Stap 2 Vind de priemontbinding van n;
Stap 3 Vind alle gemeenschappelijke priemfactoren;
Stap 4 Bereken het product van de gemeenschappelijke priemfactoren en return deze waarde;

Vraag: vinden we dit eigenlijk wel een algoritme?

Andere bekende problemen die goed algoritmisch (al dan niet efficiënt) aan te pakken zijn:

- sorteren
- zoeken
- kortste pad in een graaf
- handelsreizigersprobleem
- vierkleurenprobleem
- convex hull probleem
- stabiele huwelijken
- vier-op-een-rij, schaken, go, . . . (zijn deze wel goed aan te pakken?)
- dames op schaakbord
- torens van Hanoi
- en vele andere

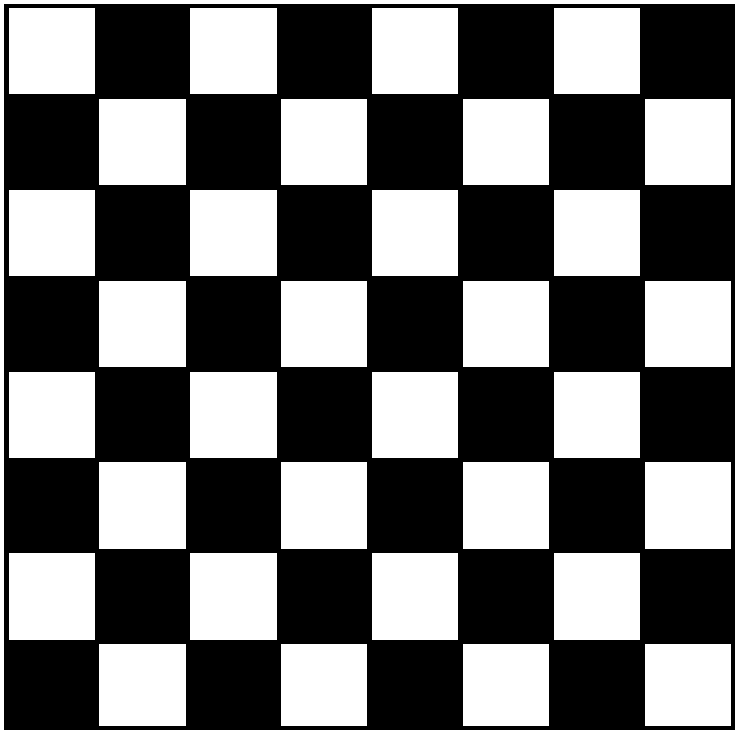
Puzzel 1:

a. Kun je een 8 bij 8 (schaak)bord geheel volleggen met dominostenen? (En zo ja, op hoeveel manieren?)

b. Idem voor een 7 bij 7 bord

c. Idem voor een 8 bij 8 bord waarbij de velden linksboven en rechtsonder zijn weggelaten

d. Idem voor een 8 bij 8 schaakbord waarbij één wit veld en één zwart veld zijn weggelaten



Een dominosteen bedekt precies twee vakjes van het schaakbord. De ogen op de steen zijn voor dit probleem niet relevant. Dit geldt ook (meestal) voor de kleur van de velden van het bord.

Puzzel 2:

$$\begin{array}{r} \mathsf{D\ O\ N\ A\ L\ D} \\ \mathsf{G\ E\ R\ A\ L\ D} \\ \hline \mathsf{R\ O\ B\ E\ R\ T} \end{array} +$$

Elke letter stelt een cijfer voor $(0, 1, \ldots, 9)$ en verschillende letters corresponderen met verschillende cijfers. Het cijfer 0 komt niet voor als meest linkse cijfer in een getal. (Dus i.h.b. is $\mathsf{D} \neq 0$, $\mathsf{G} \neq 0$ en $\mathsf{R} \neq 0$.)

Om het makkelijker te maken is gegeven dat $\mathsf{D} = 5$.

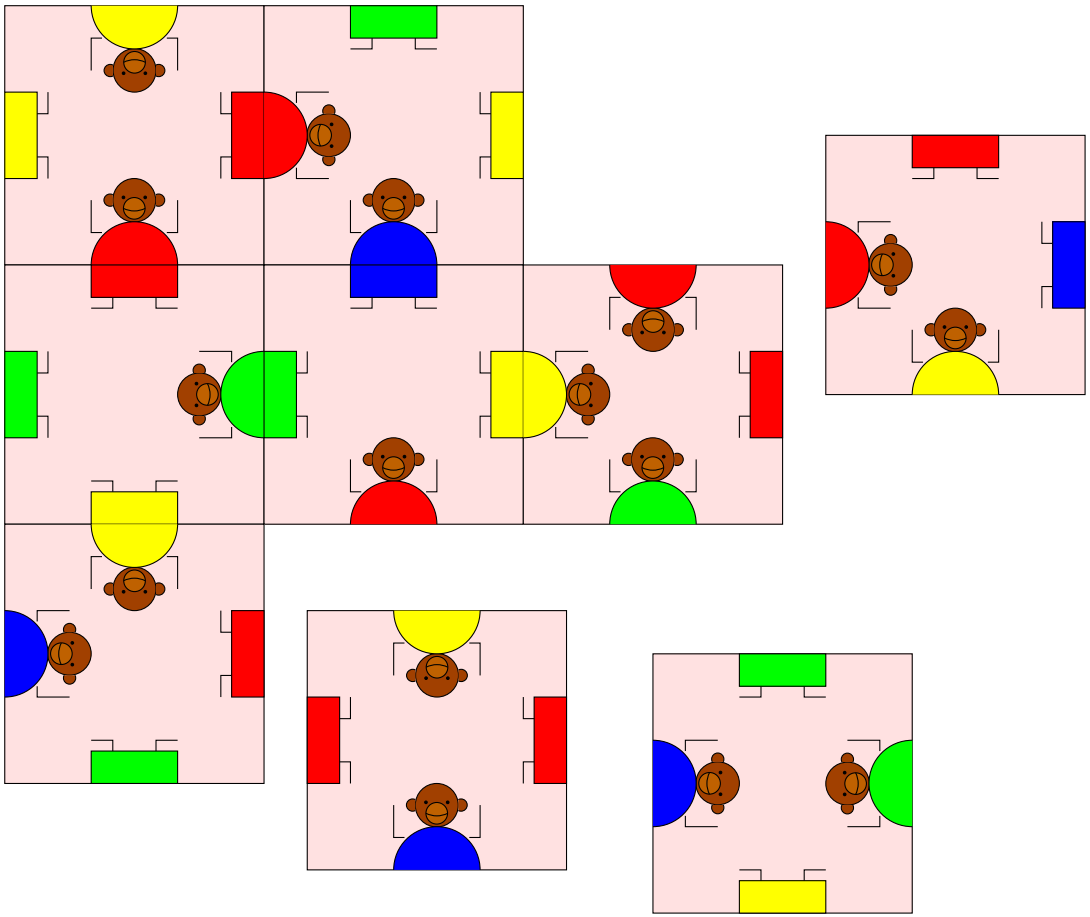
Vergelijk ook Levitin, hoofdstuk 3.4, opgave 11.

$$\begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \\ c_1 & c_2 & c_3 & c_4 & c_5 & \\ 5 & O & N & A & L & 5 \\ G & E & R & A & L & 5 \\ \hline R & O & B & E & R & T \end{array} +$$

Hierin is c_i de overdracht, dus $c_i = 0$ of 1 $(i = 1, \ldots 5)$

1. Uit kolom 6: $T = 0$ en $c_5 = 1$.
2. Uit kolom 5: $1 + 2L = R + 10 \cdot c_4 \Rightarrow$ R oneven.
3. Uit kolom 1: $c_1 + 5 + G = R \Rightarrow R \geq 5 + G > 5$,
   dus $R = 7$ of $R = 9$.
4. Uit kolom 2: $c_2 + O + E = O + 10 \cdot c_1 \Rightarrow c_2 + E = 10 \cdot c_1$.
   Als $c_1 = 0$, dan (c_2 en) $E = 0$, maar we hadden al $T = 0$.
   Gevolg: $c_1 = 1$, $E = 9$, $c_2 = 1$ en $R = 7$.
5. Doe de rest zelf.

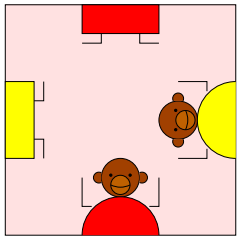
Puzzel 3: The Monkey puzzle



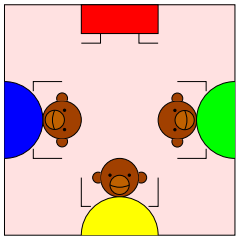
Monkey puzzle: leg de 9 stukken zo in een vierkant dat op alle aangrenzende stukken boven- en onderkant van aapjes van dezelfde kleur precies aansluiten.

Het is eenvoudig in te zien dat de puzzel van de vorige sheet opgelost kan worden. Dit is echter niet altijd het geval: het al dan niet oplosbaar zijn hangt van de stukken af.

Stel bijvoorbeeld dat we 9 dezelfde stukken hebben:



puzzel kan opgelost worden



puzzel kan niet opgelost worden

We bekijken de volgende fundamentele datastructuren:

- lineaire lijsten
- stapels en rijen
- priority queues
- grafen
- bomen

Zie Levitin 1.4 en colleges Programmeermethoden, Datastructuren.

Een lineaire lijst is een lineair geordende, eindige collectie gegevens van hetzelfde type.

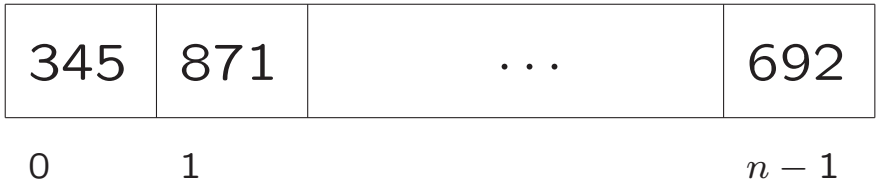
Implementatie:

- array
- pointers
  - enkelverbonden lijst
  - dubbelverbonden lijst

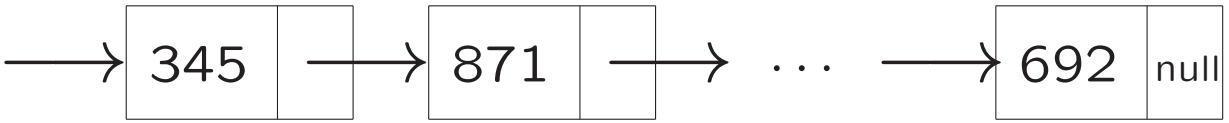
Speciale lijsten:

- stapel
- rij
- priority queue

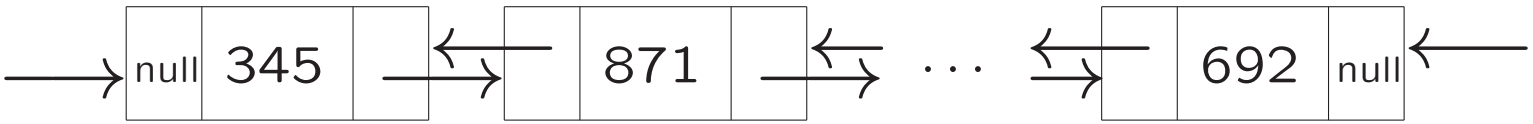
- array



- enkelverbonden lijst



- dubbelverbonden lijst

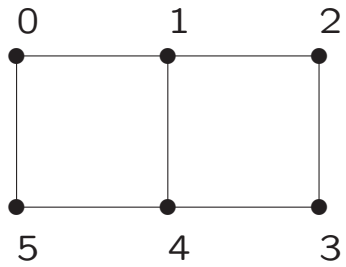


Een graaf G wordt gedefinieerd als een paar (V, E), waarbij V een eindige verzameling is van knopen (vertices) en E een verzameling van paren van knopen: de takken/pijlen (edges). Een tak/pijl (u, v) verbindt de knopen u en v met elkaar. We bekijken meestal grafen zonder lussen.

Terminologie:

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

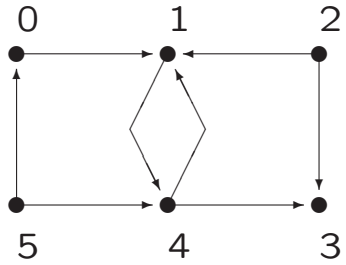
Grafen: voorbeelden



1. ongericht

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

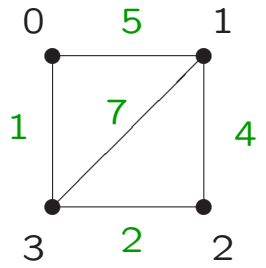
$E = \{(0, 1), (1, 2), (2, 3), (3, 4), (4, 5), (5, 0), (4, 1)\}$



2. gericht

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

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



3. ongericht en gewogen

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

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

Adjacency matrix: de **ongerichte** graaf wordt gerepresenteerd door een tweedimensionaal array `int graaf[n][n]` (n het aantal knopen), waarbij `graaf`$[i][j]$ aangeeft of er een tak tussen i en j zit.

Adjacency list: de **ongerichte** graaf wordt gerepresenteerd door een eendimensionaal array `buur* graaf[n]` (n het aantal knopen), waarbij `graaf`$[i]$ de ingang is tot een lijst van buren van knoop i.

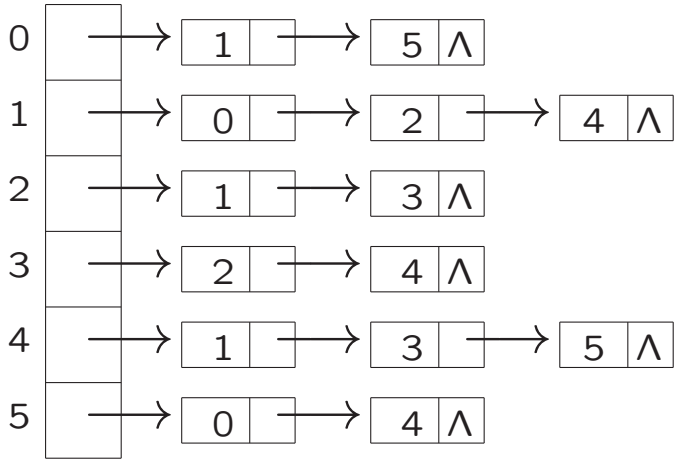
Adjacency list representatie in C++:

```
class buur { public:
  int knoopnummer;
  buur* volgende;
}
buur* graaf[n];
```

Adjacency matrix en adjacency list voor voorbeeldgraaf **1**:

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

adjacency matrix



adjacency list

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.

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.

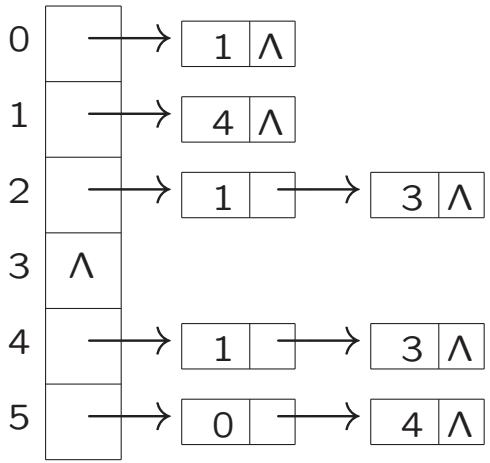
Adjacency list representatie in C++:

```
class buur { public:
  int knoopnummer;
  buur* volgende;
}
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 bepaalt of er in een gegeven ongerichte graaf hooguit twee knopen zijn met oneven graad (= aantal buren).

```
// het array graaf (buur* graaf[n]) bevat de ongerichte graaf
totaal = 0;
for ( knoop = 0; knoop < n; knoop++ ) {
    teller = 0;
    hulp = graaf[knoop];
    while ( hulp != null ) { // buurlijst van knoop aflopen
        teller++;
        hulp = hulp->volgende;
    }
    if ( teller%2 == 1 )   // oneven
        totaal++;
} // for
if ( totaal <= 2 )
    cout << " hooguit twee ... " << endl;
else
    cout << " meer dan twee ... " << endl;
```

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

  Paragrafen 1.1, 1.2, 1.4 (t/m graafrepresentaties)

- **Werkcollege:** neem je boek mee!

  donderdag 11 februari 2016, 13:45–15:30, zaal B1/B2

- **Opgaven:**

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

- zie t.z.t. de website voor de eerste programmeeropdracht

- **Volgend college:**

  vrijdag 12 februari 2016, 11:15–13:00, zaal B2