Achtste college complexiteit

2 april 2019

Polynoomevaluatie

Matrixvermenigvuldiging

Euler- en Hamiltonkringen

Zij $p(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$ een polynoom van graad $n \geq 1$, met alle a_i reële getallen ($a_i \in \mathbb{R}$).

Probleem:

Gegeven: $a_0, a_1, \ldots, a_{n-1}, a_n$ en x
Gevraagd: $p(x)$

Van links naar rechts de termen $a_i x^i$ berekenen en optellen levert een $\Theta(n^2)$-algoritme.

Als we het polynoom daarentegen van rechts naar links evalueren kunnen we eenvoudig een ordeverbetering bereiken.

Algoritme 1: "gewoon"

```
pol := a_0 + a_1 * x;  // n ≥ 1
macht := x;
for i := 2 to n do
    macht := macht * x;
    // berekent x^2, x^3, ...
    pol := pol + a_i * macht;
od
```

Basisoperatie: $*$ en $+, -$

Complexiteit: aantal $* = 2n - 1$

aantal $+, - = n$

Algoritme 2: methode van Horner

$\texttt{pol} := a_n;$
for $i := n-1$ **downto** 0 **do**
    $\texttt{pol} := \texttt{pol} * x + a_i;$
od

Gebaseerd op:

$$p(x) = ([\ldots([a_n * x + a_{n-1}] * x + a_{n-2}) * x + \ldots a_2] * x + a_1) * x + a_0$$

Complexiteit: aantal $* = n$
aantal $+, - = n$

Vraag: kan het met minder $*$ en $+, -$?
Antwoord: nee !

Algoritmen gebaseerd op het doen van vergelijkingen konden we beschrijven met **beslissingsbomen**.
Een model om rekenkundige algoritmen (algoritmen die gebaseerd zijn op $+, -, *$ en $/$) mee te beschrijven: schema's.
Een schema

- is een eindige serie stappen van de vorm $s_i := q \circ r$;
- hierin is $\circ$ een rekenkundige operatie: $*, /, +$ of $-$
- q en r zijn constanten (bijvoorbeeld $1, -1, \pi^2, \ldots$) of invoerwaarden (hier a_k's of x) of tussenresultaten van eerdere stappen
- de laatste stap uit het schema berekent het eindresultaat (hier dus $p(x)$)

Horner met $n = 2$: $s_1 := a_2 * x$; $s_2 := s_1 + a_1$; $s_3 := s_2 * x$; $s_4 := s_3 + a_0$;

Stelling. Elk schema (dat alleen $+, -$ en $*$ gebruikt) om $a_0 + a_1x + a_2x^2 + \ldots + a_nx^n$ te berekenen moet ten minste n $(+,-)$-stappen doen en n $*$-stappen.
Bewijs voor $(+,-)$-stappen volgt (vervang x door 1) uit:

Lemma. Een schema (dat alleen $+, -$ en $*$ gebruikt) om $a_0 + a_1 + a_2 + \ldots + a_{n-1} + a_n$ te berekenen moet ten minste n $(+,-)$-stappen hebben.

Het bewijs gaat met inductie naar n, met $n = 0$ als flauw basisgeval. Vervang overal a_n door 0. Dit geeft een schema dat $a_0 + a_1 + a_2 + \ldots + a_{n-1}$ berekent. Kijk naar de eerste $(+,-)$-stap die a_n gebruikt (die bestaat zeker): $s_i := q \pm 0$ of $s_i := 0 + r$ of $s_i := 0 - r$; laat de twee eerste weg en vervang overal s_i door q danwel r, of $s_i := -1 * r$ vervangt zonodig de derde. We hebben zo een $(+,-)$-stap geëlimineerd uit het schema. Gebruik nu inductie.

De methode van Horner berekent $p(x)$ met n vermenigvuldigingen en n optellingen $(+/-)$. Er bestaat geen algoritme dat het probleem voor algemene p en x met minder $*$'s en $+/-$'s kan oplossen. De methode van Horner is derhalve optimaal.*

Maar misschien kan het wel beter voor polynomen die een heel speciale vorm hebben.

*de schets van het bewijs van de ondergrens (vorige twee slides) is geen tentamenstof

Polynoomevaluatie met preprocessing: bewerk het polynoom tot een polynoom in een speciale vorm waarop een nieuw evaluatie-algoritme sneller werkt.

Het polynoom

Laat $p(x) = x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$, met $n = 2^k - 1$.

$p(x)$ is dus een monisch polynoom, dat wil zeggen dat $a_n = 1$. We kunnen zonder beperking der algemeenheid aannemen dat het polynoom dat we willen evalueren monisch is. Hetzelfde geldt voor de aanname dat $n = 2^k - 1$.

De speciale vorm (*recursief* geformuleerd):

$$p(x) = (x^j + b) * q(x) + r(x),$$

waarin q en r ook weer monisch zijn en in de speciale vorm staan, beide van graad $2^{k-1} - 1$ zijn, en $j = 2^{k-1}$.

Voorbeeld (met $n = 7$)

$$p(x) = x^7 + 6x^6 + 5x^5 + 4x^4 + 3x^3 + 2x^2 + x + 1 =$$

$$(x^4+2)[(x^2+4)(x+6)+(x-20)]+[(x^2-10)(x-10)+(x-107)]$$

Om dit polynoom in x te evalueren gebruikt Horner 7 $*$'s en 7 $+/-$'s, maar het kan met 5 $*$'s en 10 $+/-$'s.

Een gegeven monisch polynoom p van graad $n = 2^k - 1$ is eenvoudig om te zetten naar de speciale vorm.

We willen p schrijven als: $p(x) = (x^j + b) * q(x) + r(x)$ met q en r monisch, beide van graad $2^{k-1} - 1$, en $j = 2^{k-1}$. De waarde van b en de coëfficiënten van q en r zijn hieruit simpel af te lezen.
Immers: als $q(x) = x^{j-1} + q_{j-2}x^{j-2} + \ldots + q_0$ en $r(x) = x^{j-1} + r_{j-2}x^{j-2} + \ldots + r_0$, dan geldt:

$$b + 1 = a_{j-1}, q_\ell = a_{\ell+j}, b * q_\ell + r_\ell = a_\ell,$$

voor $\ell = 0, 1, \ldots, j-2$ en de a_i de coëfficiënten van p.

Vervolgens kunnen q en r op dezelfde manier in de speciale vorm gebracht worden, etc. Algoritme met complexiteit: zie opgave 48.

Als het polynoom p in de juiste vorm staat kan $p(x)$ als volgt geëvalueerd worden:

1. Evalueer $q(x)$ en $r(x)$ recursief

2. Bereken de x^j's: nodig hiervoor zijn $x, x^2, x^4, \ldots, x^{2^{k-1}}$. Bereken deze alle van tevoren: $k-1$ $*$'s

3. Vermenigvuldig $(x^j + b)$ met $q(x)$ en tel er $r(x)$ bij op: 1 $*$ en 2 $+/-$'s

Zij $M(k)$ = het aantal $*$'s dat gedaan wordt om een monisch polynoom (in de speciale vorm) van graad $2^k - 1$ te evalueren, zonder de berekening van de x^j mee te tellen.

Dan voldoet $M(k)$ aan de volgende recurrente betrekking:

$$M(k) = \begin{cases} 0 & k = 1 \\ 2M(k-1) + 1 & k > 1 \end{cases}$$

Oplossing: $M(k) = 2^{k-1} - 1 \longrightarrow \frac{n-1}{2}$

Aantal $+/-$'s

Zij $A(k)$ = het aantal $+/-$'s dat gedaan wordt om een monisch polynoom (in de speciale vorm) van graad $2^k - 1$ te evalueren.

Dan voldoet $A(k)$ aan de volgende recurrente betrekking:

$$A(k) = \begin{cases} 1 & k = 1 \\ 2A(k-1) + 2 & k > 1 \end{cases}$$

Oplossing: $A(k) = 3 * 2^{k-1} - 2 \longrightarrow \frac{3n-1}{2}$

Zij A en B twee $n \times n$ matrices met elementen a_{ij} en b_{ij} ($1 \leq i, j \leq n$). De elementen c_{ij} van het (matrix-)product $C = A \cdot B$ zijn dan als volgt gedefinieerd: $c_{ij} = \sum_{k=1}^{n} a_{ik} * b_{kj}$

Een standaard $\Theta(n^3)$ algoritme hiervoor is recht-toe recht-aan uit definitie:

for $i := 1$ **to** n **do**
  for $j := 1$ **to** n **do**
    $c_{ij} := 0$;
    for $k := 1$ **to** n **do**
      $c_{ij} := c_{ij} + a_{ik} * b_{kj}$;
    od
  od
od

Voorbeeld: product van twee 2×2 matrices algemeen.

$$\underbrace{\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}}_{B} = \underbrace{\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}}_{C}$$

$$c_{11} = a_{11} * b_{11} + a_{12} * b_{21}$$
$$c_{12} = a_{11} * b_{12} + a_{12} * b_{22}$$
$$c_{21} = a_{21} * b_{11} + a_{22} * b_{21}$$
$$c_{22} = a_{21} * b_{12} + a_{22} * b_{22}$$

Het product van twee matrices is overigens ook gedefinieerd voor niet-vierkante matrices A en B, mits maar geldt dat aantal kolommen van A = aantal rijen van B.

Voorbeeld: product van twee 2×2 matrices.
We berekenen c_{21}

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$$

A B C

via $c_{21} = a_{21} * b_{11} + a_{22} * b_{21}$

We kunnen c_{21} trouwens ook berekenen als: $(a_{21} + a_{22}) * b_{11} + a_{22} * (b_{21} - b_{11})$. Dit lijkt weinig zinvol, maar toch ...

Voorbeeld 2×2: aantal $*$

In totaal kost de recht-toe recht-aan methode (links) voor dit voorbeeld 8 vermenigvuldigingen van array-elementen. Het kan echter door 'herschrijven' met 7 (rechts):

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$$

$$\begin{aligned} 19 &= 1*5+2*7 \\ 22 &= 1*6+2*8 \\ 43 &= 3*5+4*7 \\ 50 &= 3*6+4*8 \end{aligned}$$

$$\begin{aligned} M_1 &= (1+4)*(5+8) = 65 \\ M_2 &= (3+4)*5 = 35 \\ M_3 &= 1*(6-8) = -2 \\ M_4 &= 4*(7-5) = 8 \\ M_5 &= (1+2)*8 = 24 \\ M_6 &= (3-1)*(5+6) = 22 \\ M_7 &= (2-4)*(7+8) = -30 \\ 19 &= M_1+M_4-M_5+M_7 \\ 22 &= M_3+M_5 \\ 43 &= M_2+M_4 \\ 50 &= M_1-M_2+M_3+M_6 \end{aligned}$$

Algemeen: $M_2 = (a_{21}+a_{22})*b_{11}$; $M_4 = a_{22}*(b_{21}-b_{11})$; dan $M_2+M_4 = c_{21}$, etc.

Neem aan dat $n = 2^k$. We kunnen dan de matrices A, B en C ieder opsplitsen in vier $\frac{n}{2} \times \frac{n}{2}$ deelmatrices:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

Terzijde: de complexiteit van matrixvermenigvulding is in $\Omega(n^2)$, want ieder algoritme moet in ieder geval de $n \times n$ matrices A en B lezen en het resultaat (alle c_{ij}) berekenen/schrijven.

We kunnen $C = A \cdot B$ recursief berekenen via*:

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

met
$$\begin{aligned} C_{11} &= A_{11} \cdot B_{11} + A_{12} \cdot B_{21} \\ C_{12} &= A_{11} \cdot B_{12} + A_{12} \cdot B_{22} \\ C_{21} &= A_{21} \cdot B_{11} + A_{22} \cdot B_{21} \\ C_{22} &= A_{21} \cdot B_{12} + A_{22} \cdot B_{22} \end{aligned}$$

Dit zijn 8 vermenigvuldigingen en 4 optellingen van $\frac{n}{2} \times \frac{n}{2}$ matrices. Voor $M(k)$, het aantal vermenigvuldigingen van array-elementen, geldt: $M(k) = 8M(k-1)$ en $M(0) = 1$, wat $M(k) = 8^k = n^3$ geeft. Hetzelfde als de "gewone" methode.

*Ga voor $n = 4$ na dat dit correct is door een en ander uit te schrijven

Analoog aan het 2×2 geval kunnen we de berekening van de vier C_{ij}'s herschrijven zodanig dat we nog maar 7 vermenigvuldigingen van $\frac{n}{2} \times \frac{n}{2}$ matrices hoeven te doen.

$S_1, \ldots, S_{10}$ zijn $\frac{n}{2} \times \frac{n}{2}$ matrices die alle de som of het verschil van twee deelmatrices van A en B zijn:

$$
\begin{aligned}
S_1 &= B_{12} - B_{22} \\
S_2 &= A_{11} + A_{12} \\
S_3 &= A_{21} + A_{22} \\
S_4 &= B_{21} - B_{11} \\
S_5 &= A_{11} + A_{22} \\
S_6 &= B_{11} + B_{22} \\
S_7 &= A_{12} - A_{22} \\
S_8 &= B_{21} + B_{22} \\
S_9 &= A_{11} - A_{21} \\
S_{10} &= B_{11} - B_{12}
\end{aligned}
$$

$P_1, \ldots, P_7$ zijn $\frac{n}{2} \times \frac{n}{2}$ matrices die alle het product van twee matrices S en/of deelmatrices van A en B zijn:

$$
\begin{array}{lll}
P_1 & = A_{11} \cdot S_1 & = A_{11} \cdot B_{12} - A_{11} \cdot B_{22} \\
P_2 & = S_2 \cdot B_{22} & = A_{11} \cdot B_{22} + A_{12} \cdot B_{22} \\
P_3 & = S_3 \cdot B_{11} & = A_{21} \cdot B_{11} + A_{22} \cdot B_{11} \\
P_4 & = A_{22} \cdot S_4 & = A_{22} \cdot B_{21} - A_{22} \cdot B_{11} \\
P_5 & = S_5 \cdot S_6 & = A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22} \\
P_6 & = S_7 \cdot S_8 & = A_{12} \cdot B_{21} + A_{12} \cdot B_{22} - A_{22} \cdot B_{21} - A_{22} \cdot B_{22} \\
P_7 & = S_9 \cdot S_{10} & = A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12}
\end{array}
$$

Merk op dat de berekening in de rechterkolom alleen voor onze informatie is, de enige *feitelijke* matrixvermenigvuldigingen staan in de middenkolom! Dit zijn er 7.

De matrices C_{11}, C_{12}, C_{21} en C_{22} kunnen nu als volgt bepaald worden:

$$\begin{array}{lll} C_{11} & = P_5 + P_4 - P_2 + P_6 & = A_{11} \cdot B_{11} + A_{12} \cdot B_{21} \\ C_{12} & = P_1 + P_2 & = A_{11} \cdot B_{12} + A_{12} \cdot B_{22} \\ C_{21} & = P_3 + P_4 & = A_{21} \cdot B_{11} + A_{22} \cdot B_{21} \\ C_{22} & = P_5 + P_1 - P_3 - P_7 & = A_{21} \cdot B_{12} + A_{22} \cdot B_{22} \end{array}$$

We krijgen dus zeven keer een (recursieve) matrixvermenigvuldiging van $\frac{n}{2} \times \frac{n}{2}$ matrices, en we doen achttien optellingen van $\frac{n}{2} \times \frac{n}{2}$ matrices per recursiestap.

Denk er aan dat matrixvermenigvuldiging niet commutatief is: doorgaans geldt $A \cdot B \neq B \cdot A$. Merk op dat bij het herschrijven op deze en de vorige twee slides nergens stiekem $X \cdot Y$ door $Y \cdot X$ is vervangen; alle gelijkheden zijn dan ook geldig. Controleer zelf dat het allemaal klopt.

Dit levert de volgende recurrente betrekkingen op voor het aantal vermenigvuldigingen van array-elementen $M(k)$, respectievelijk het aantal $+/-$ van array-elementen $A(k)$, met $n = 2^k$:

$$M(k) = \begin{cases} 1 & k = 0 \\ 7M(k-1) & k > 0 \end{cases}$$

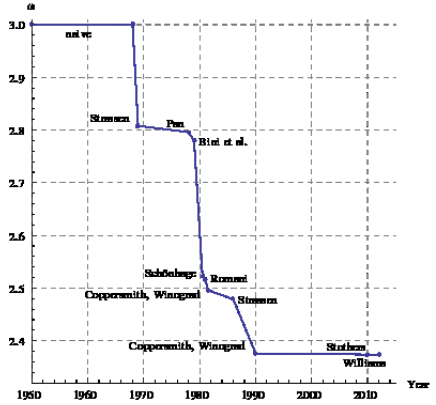
$$A(k) = \begin{cases} 0 & k = 0 \\ 7A(k-1) + 18 \cdot (2^{k-1})^2 & k > 0 \end{cases}$$

Oplossing: $M(k) = 7^k = 2^{\lg 7^k} = 2^{k \cdot \lg 7} = n^{\lg 7} \approx n^{2,81}$
$A(k) = 6 \cdot 7^k - 6 \cdot 4^k \in \Theta(n^{\lg 7})$.

We hebben dus: Strassen (1969) met $\Theta(n^{2,81})$

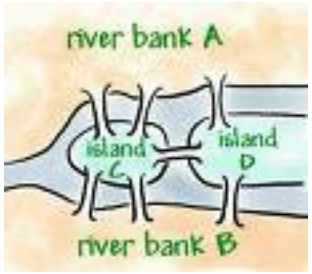
Het kan nog sneller!

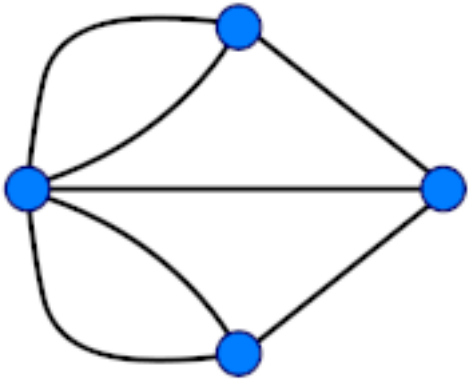
Pan (1978):	$\Theta(n^{2,796})$
Coppersmith-Winograd (1990):	$\Theta(n^{2,376})$
Stothers (2010):	$\Theta(n^{2,3737})$
Williams (2012):	$\Theta(n^{2,3729})$
Le Gall (2014):	$\Theta(n^{2,3728639})$



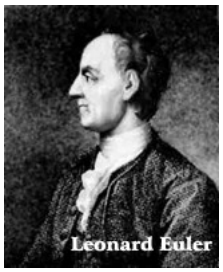
Echter dit soort algoritmen heeft zulke grote constanten verstopt in de Θ dat ze alleen asymptotisch sneller zijn dan Strassen voor matrices die te groot zijn om met de huidige hardware te berekenen. Vermoeden: $\Theta(n^{2+\epsilon})$ voor elke $\epsilon > 0$.

Koningsberger bruggenprobleem:





Kun je een wandeling door de stad maken waarbij je elke brug precies één keer beloopt en je weer terugkeert in het beginpunt?



Leonard Euler, 1736

Gegeven een samenhangende, ongerichte graaf $\mathcal{G} = (V, E)$.

- Een pad van u naar v is een rij knopen waarvoor geldt dat tussen elk tweetal opeenvolgende knopen uit die rij een tak (lijn) zit.
- De lengte van een pad $=$ het aantal takken op dat pad $=$ het aantal knopen $-\,1$.
- Een kring in een ongerichte graaf is een pad dat begint en eindigt in dezelfde knoop (een *gesloten* pad dus) en dat geen enkele tak meer dan één keer bevat*. Een kring bestaat dus uit allemaal verschillende takken.
- Merk op: een gesloten pad dat geen enkele *knoop* meer dan één keer bevat† is een speciaal geval van een kring. Zo'n pad bestaat dus uit allemaal verschillende knopen.

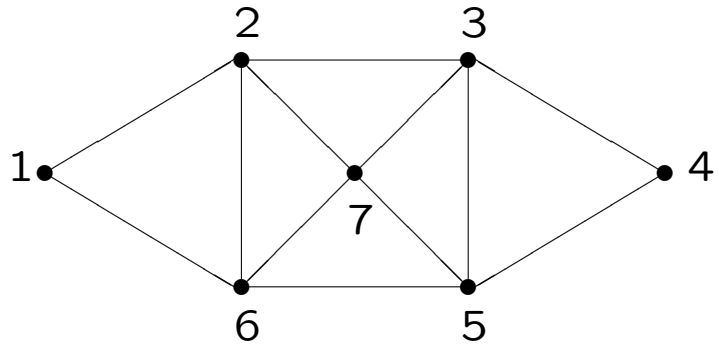
*Bij Fundamentele informatica 1 noemden we dit een circuit.

†Bij Fundamentele informatica 1 noemden we dit een cykel

Gegeven een samenhangende, ongerichte graaf $\mathcal{G} = (V, E)$.

Definitie: een kring in $\mathcal{G}$ die alle takken van $\mathcal{G}$ bevat heet een Eulerkring. Deze bevat dus *elke* tak precies 1 keer.

Voorbeeld:



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

Eulerkringprobleem. Gegeven een samenhangende ongerichte graaf $\mathcal{G} = (V, E)$. Heeft $\mathcal{G}$ een Eulerkring?

Dit is een voorbeeld van een beslissingsprobleem: het antwoord is ja of nee.

Stelling

Een samenhangende ongerichte graaf heeft een Eulerkring $\Longleftrightarrow$ de graad van iedere knoop is even.

Het is dus heel gemakkelijk (= polynomiaal) na te gaan of een graaf een Eulerkring heeft: $O(|E|)$.

Opmerking

Het bewijs van "$\Longleftarrow$" is constructief: het geeft je meteen een (overigens $O(|E| + |V|)$) algoritme om zo'n Eulerkring te vinden.

Algoritme voor de constructie van een Eulerkring:

1. Controleer de samenhangendheid en controleer of elke knoop even graad heeft;

2. **if** controle positief **then**

    3. Kies een knoop v;

    4. Construeer een kring $\mathcal{C}$ (van v naar v);
    // gewoon lopen en steeds andere, nog ongebruikte,
    // takken bewandelen tot je in v terug bent

5. **while** er nog onbewandelde takken zijn **do**

   6. Zoek/onthoud een knoop w van $\mathcal{C}$ die nog onbewandelde takken heeft;
      // kan dezelfde knoop zijn als in de vorige ronde

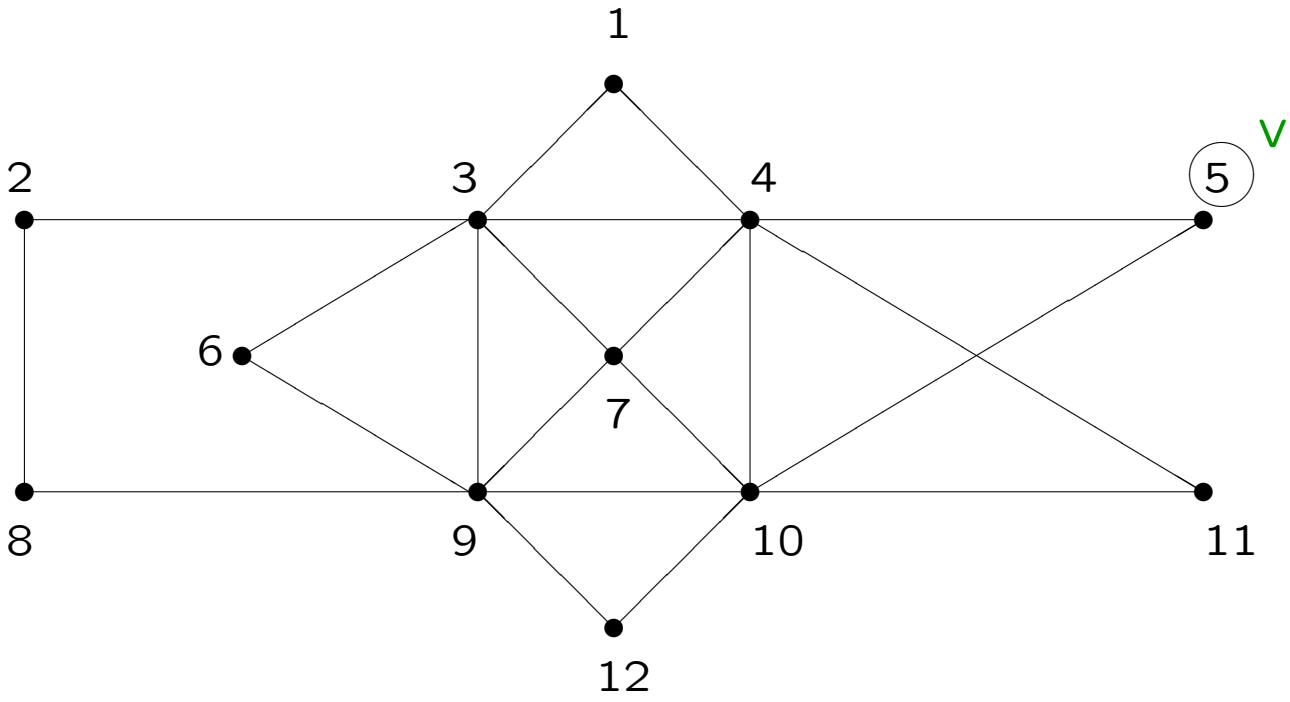
   7. Construeer een kring (van w naar w) bestaande uit ongebruikte takken;

   8. Voeg deze kring in in $\mathcal{C}$ op de plek van w;

9. **od**

10. **fi**

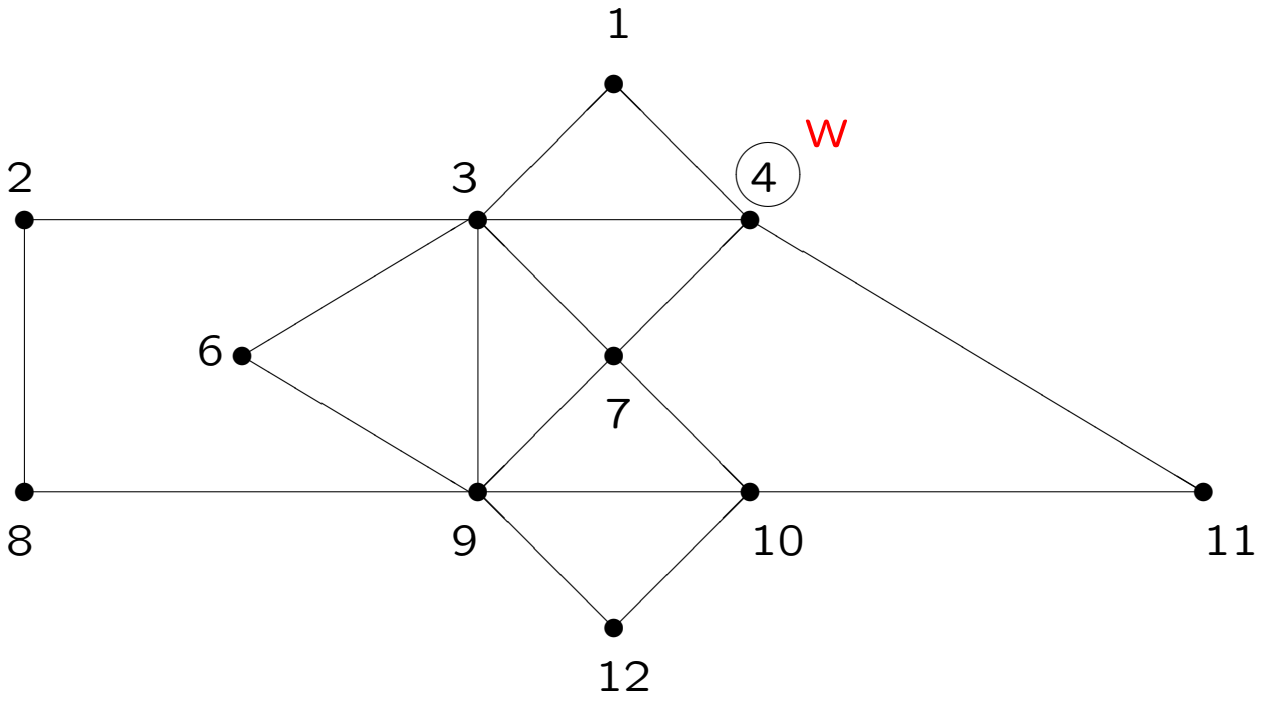
Opbouw Eulerkring 1



Kring vanuit v: $5, 4, 10, 5$

$C = 5, 4, 10, 5$

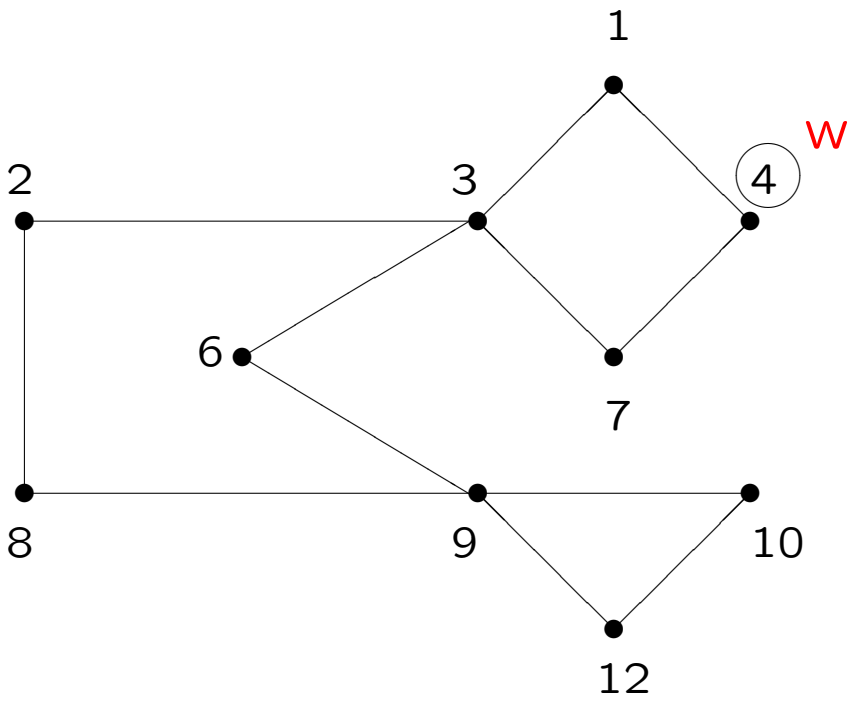
Opbouw Eulerkring 2



Kring vanuit w: $4, 11, 10, 7, 9, 3, 4$

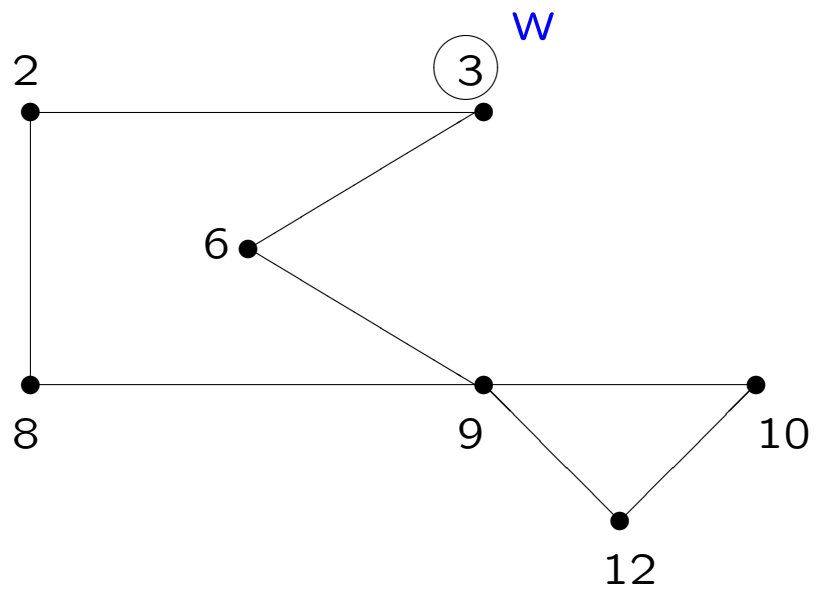
$C = 5,\ 4, 11, 10, 7, 9, 3, 4,\ 10, 5$

Opbouw Eulerkring $2\frac{1}{2}$



Kring vanuit w: $4, 1, 3, 7, 4$

$C = 5,\ 4, 1, 3, 7, 4, 11, 10, 7, 9, 3, 4,\ 10, 5$



Kring vanuit w: $3, 2, 8, 9, 10, 12, 9, 6, 3$
(of eerst $3, 2, 8, 9, 6, 3$ en dan $9, 10, 12$)

$C = 5,\ 4, 1,\ 3, 2, 8, 9, 10, 12, 9, 6, 3,\ 7, 4, 11, 10, 7, 9, 3, 4,\ 10, 5$

Complexiteit van dit algoritme: $O(|V| + |E|)$

Stap 1: samenhangendheid controleren met behulp van DFS en tegelijk de graden bepalen: $O(|V| + |E|)$.

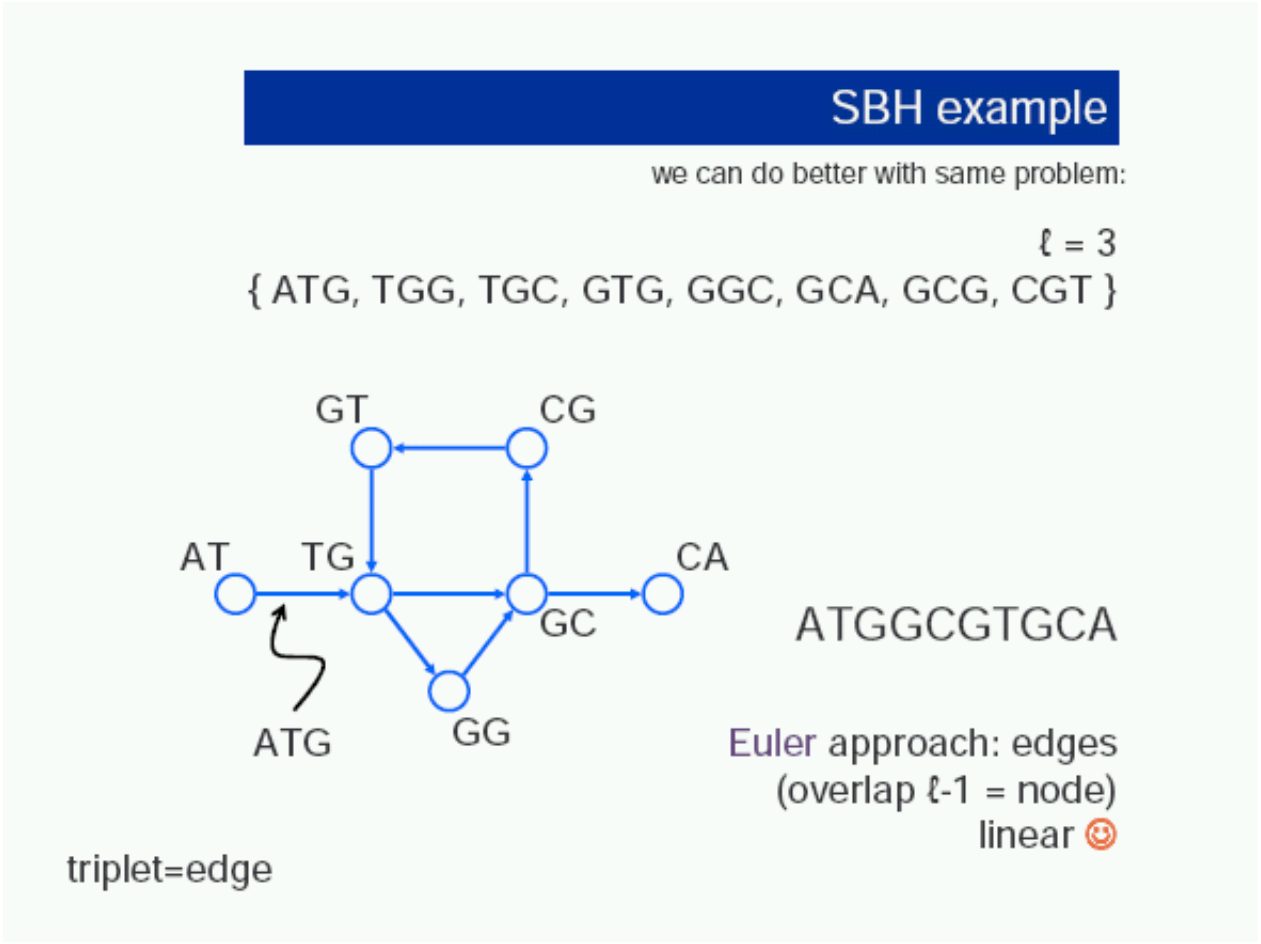
Stap 4 t/m 8: kan in $O(|E|)$ stappen

- gebruik een kopie van $\mathcal{G}$ ($O(|V| + |E|)$ om die te maken).
- haal tijdens de constructie van $\mathcal{C}$ een tak weg (uit de kopie) zodra die gebruikt is.
- houd de kring $\mathcal{C}$ bij als enkelverbonden lijst, met een pointer naar de eerste knoop die nog onbewandelde takken heeft.
  We kiezen in Stap 6. dus de eerste de beste knoop op $\mathcal{C}$ die nog onbewandelde takken heeft.

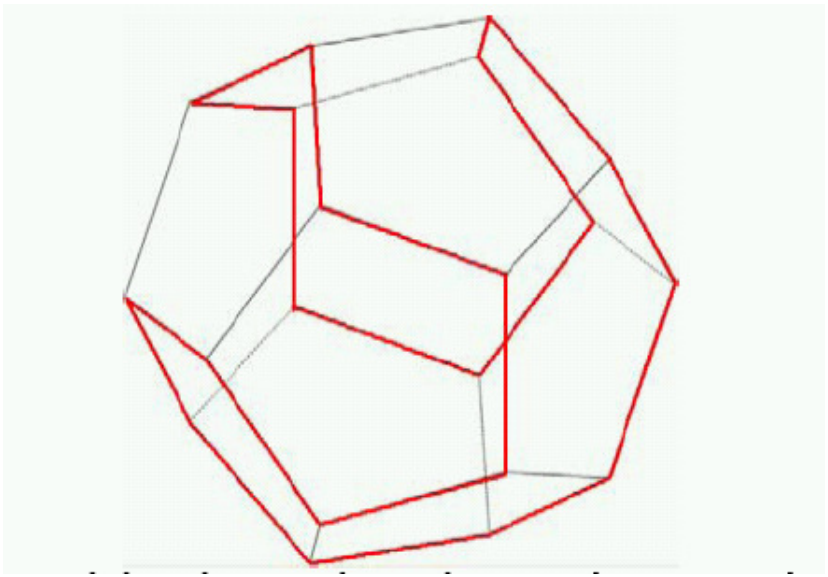
Opmerkingen bij implementatie en complexiteit:

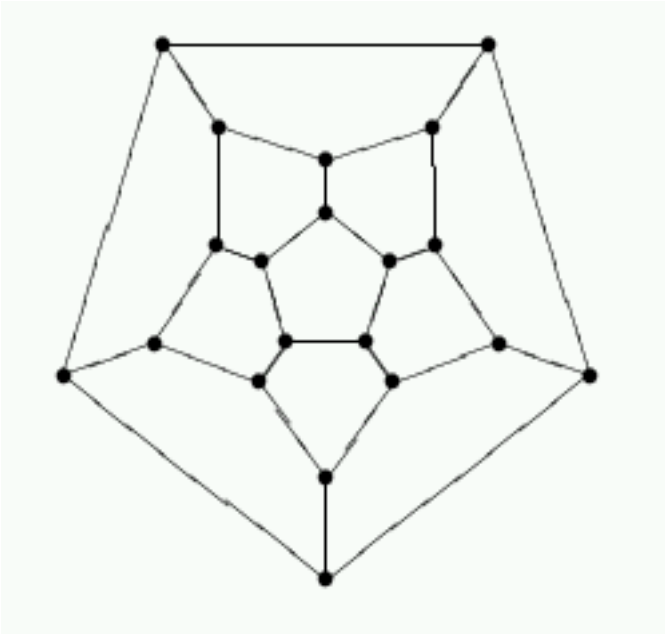
- gebruik een adjacencylist als datastructuur en laat de buurlijsten oplopend gesorteerd zijn
- als een tak (v, w) gebruikt is halen we alleen w uit de buurlijst van v, maar niet andersom (dat komt later)
- de twee implementatie-opmerkingen hierboven zijn bedoeld om complexiteit $O(|E|)$ te verkrijgen in plaats van $O(|E|^2)$
- zie opgave 50
- zie opgave 52 voor een alternatief (langzamer) algoritme (van Fleury, 1883)
- om de complexiteit te beschrijven hebben we verschillende soorten operaties geteld en op één hoop gegooid

Uit: Molecular Computational Biology (oud mastercollege)



Kun je een wandeling door de graaf rechtsonder maken waarbij elke *knoop* precies één keer bezocht wordt en je weer in het startpunt eindigt?





Sir William Rowan Hamilton, 1859

Gegeven een ongerichte (of gerichte) graaf $\mathcal{G} = (V, E)$.

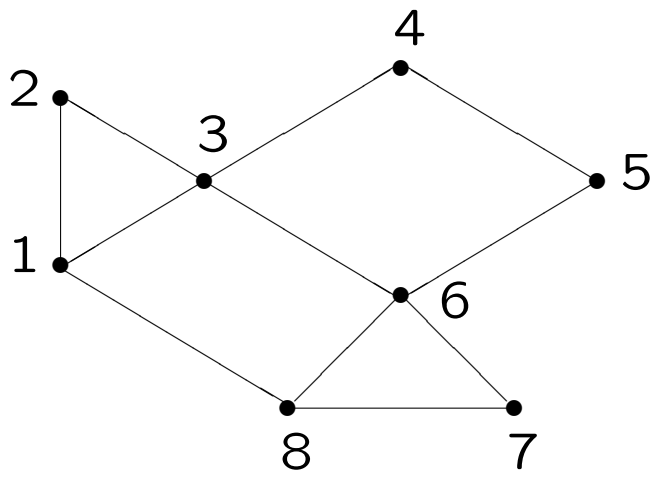
Definitie: een Hamiltonkring in $\mathcal{G}$ is een kring die elke knoop precies één keer bevat.

Bijbehorend beslissingsprobleem: Hamiltonkringprobleem (HC) voor ongerichte* grafen. Gegeven een ongerichte graaf $\mathcal{G} = (V, E)$. Heeft $\mathcal{G}$ een Hamiltonkring?

Naamgeving: Als een graaf $\mathcal{G}$ een Hamiltonkring heeft spreken we van een ja-instantie van het probleem. Als zo'n kring niet bestaat heet $\mathcal{G}$ een nee-instantie.

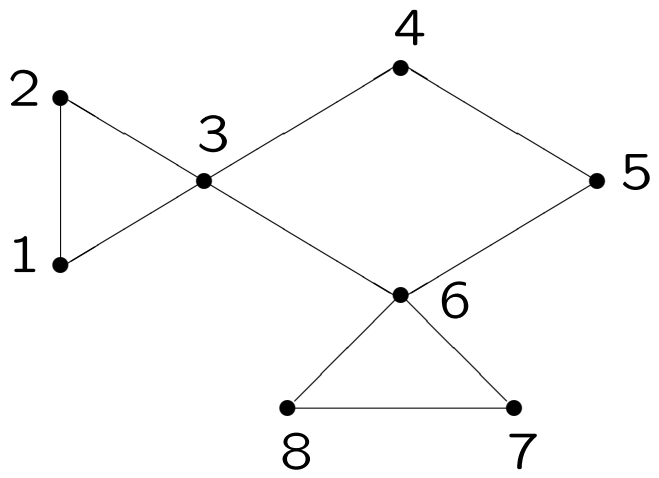
* analoog voor gerichte grafen

Voorbeeld 1:



Deze graaf heeft een Hamiltonkring, namelijk: 1, 2, 3, 4, 5, 6, 7, 8

Voorbeeld 2:



Deze graaf heeft geen Hamiltonkring, maar wel een Hamiltonpad, namelijk: 1, 2, 3, 4, 5, 6, 7, 8.

Vermomde Hamiltonproblemen:

1. Kan een paard over een n bij n schaakbord een serie sprongen maken zodanig dat hij alle n^2 velden precies één keer bezoekt en ten slotte weer terugkeert in zijn beginveld?

2. Gegeven een groep van n mensen. Van elk tweetal is bekend of ze elkaar wel of niet kennen. Kunnen deze mensen zo aan een ronde tafel worden geplaatst, dat iedereen twee bekenden als buren heeft?

3. (Gray-code.) Er zijn 2^n binaire getallen van n bits. Kunnen deze zo (cyclisch) achter elkaar worden geplaatst dat opeenvolgende getallen slechts op één plaats verschillen?

1. Een exponentieel algoritme is snel gevonden: genereer alle $n!$ mogelijke Hamiltonkringen (n is aantal knopen van $\mathcal{G}$) en controleer daarvan of het een Hamiltonkring is.
2. Het *controleren* of een kandidaat Hamiltonkring echt een Hamiltonkring is, *is polynomiaal*.
3. Het *vinden* van een Hamiltonkring daarentegen is/lijkt *extreem moeilijk* (exponentieel).
4. Er is geen polynomiaal algoritme voor dit probleem bekend; zelfs niet voor het beslissingsprobleem HC, dat alleen vraagt *of* $\mathcal{G}$ een Hamiltonkring heeft.
5. Er is ook niet bewezen dat een exponentieel algoritme nodig is om HC op te lossen.

6. Als $\mathcal{G}$ een ja-instantie is van HC, dus een Hamiltonkring heeft, is er een eenvoudige (= polynomiale) manier om dat aan te tonen met behulp van de juiste hint (certificaat). Immers, in dit geval is het certificaat zo'n Hamiltonkring: controle of deze echt een Hamiltonkring is, is polynomiaal.

7. De enige manier om te laten zien dat $\mathcal{G}$ een nee-instantie is, dus geen Hamiltonkring bevat, lijkt: som alle $n!$ kandidaat Hamiltonkringen op en laat zien dat ze geen Hamiltonkring zijn. Dat is exponentieel.

8. Er zijn eenvoudige en lastige instanties . . . $\rightarrow$ NP-volledigheid

- Volgende college:
  dinsdag 9 april, 11.00 – 12.45, zaal 174

- Eerstvolgende werkcollege:
  dinsdag 2 april, 13.30 – 15.15, zaal 174
  Opgaven 31, 33, 29, 34, 35

- Derde huiswerkopgave (van de vier):

  * deadline: dinsdag 23 april; LaTeX; print $\rightarrow$ college

  * www.liacs.leidenuniv.nl/~graafjmde/COMP/