Grafen

Classic Computer Science Algorithms

Stijn Lievens Pieter-Jan Maenhaut Koen Mertens Lieven Smits

AJ 2021–2022

HO GENT

Motivatie

In heel wat praktische situaties heeft men te maken met de situatie waarin 'objecten' verbonden zijn door een bepaalde relatie:

- steden zijn met elkaar verbonden m.b.v. wegen; kost: de afstand in kilometer,
- computers zijn verbonden m.b.v. netwerkkabels; kost: de communicatiesnelheid van de verbinding,
- luchthavens zijn met elkaar verbonden door directe vluchten; kost: de duur van de rechtstreekse vlucht.

HO GENT

Definitie

Definitie

Een GRAAF G *bestaat uit een verzameling* KNOPEN V*, en een verzameling* BOGEN E*. Elke boog verbindt twee knopen, en we noteren* $e = (v, w)$*. De graaf* G *wordt genoteerd als het koppel* (V, E)*, dus* $G = (V, E)$*.*

Opmerking:

- als $(v, w) \in E$, dan zijn v en w ADJACENT; v en w zijn INCIDENT met e (en omgekeerd)
- De BUREN van een knoop v zijn alle knopen w die adjacent zijn met v;
- aantal buren van een knoop: GRAAD van die knoop
- $\#V$, wordt de ORDE van de graaf genoemd; notatie n
- $\#E$, noemt men de GROOTTE; notatie m

HO GENT

Gerichte versus ongerichte graaf

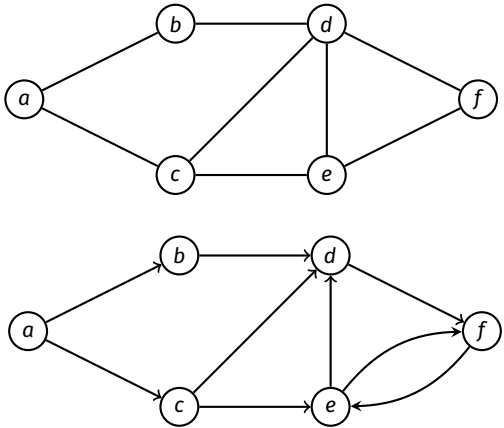
- Wanneer de boogparen niet geordend zijn dan spreekt men van een ONGERICHTE graaf.
- Wanneer de boogparen wel geordend zijn dan heeft men een GERICHTE graaf.
  In een gerichte graaf heeft een boog (v, w) een STAART v en een KOP w.
- GEWOGEN graaf: associeer getal met de bogen

Voorbeelden

- vriendschapsgraaf Facebook: ongericht
- volgersgraaf Twitter: gericht

HO GENT

Grafische voorstelling



HO GENT

Paden en cykels

Definitie

Een PAD in een graaf G is een opsomming van knopen $(v_1, v_2, \ldots, v_k)$ zodanig dat er een boog bestaat tussen v_i en v_{i+1} voor $i \in \{1, 2, \ldots, k-1\}$. De LENGTE van dit pad is $k-1$, zijnde het aantal bogen op dit pad.

Opmerking:
$k = 1$ is toegestaan: (v) is een pad van lengte nul.

Definitie

Een ENKELVOUDIGE CYKEL in een graaf is een pad waarvan de lengte strikt positief is en dat begint en eindigt in dezelfde knoop, waarbij alle knopen (behalve de start- en eindknoop) verschillend zijn en waarbij bovendien geen boog méér dan een keer wordt doorlopen.

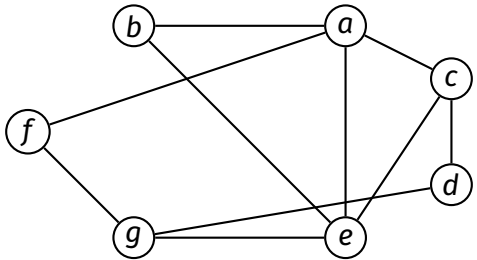
HO GENT

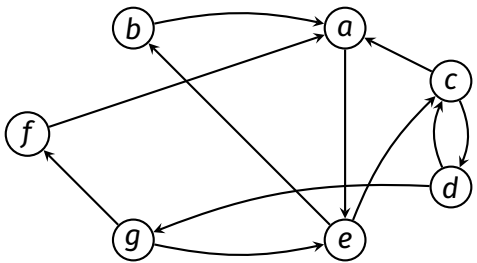
Voorbeeld

- In de ongerichte graaf is (a, b, d, c, e, f) een pad van lengte 5 van a naar f.
  Dit is *geen* pad in de gerichte graaf omdat er geen boog is van d naar c (enkel van c naar d).
- In beide grafen is het pad (d, f, e, d) een enkelvoudige cykel van lengte 3.
- In de gerichte graaf is (e, f, e) een enkelvoudige cykel van lengte 2, maar dit is *niet* zo in de ongerichte graaf omdat we de boog (e, f) twee keer zouden gebruiken in dit pad.
- In de ongerichte graaf is (a, b, d, f, e, d, c, a) een pad van lengte 7 van a naar a.
  Dit pad is *geen* enkelvoudige cykel want de knoop d wordt twee keer gebruikt!

HO GENT

Oefeningen





HO GENT

Oefeningen

1. Beschouw de gerichte en ongerichte graaf op vorige slide (Figuur 5.3 in de cursus)
   1.1 Geef de bogenverzameling van deze twee grafen.
   1.2 Geef voor beide grafen de volgende verzameling $\textbf{buren}(e)$. Wat is de graad van de knoop e in beide gevallen?
   1.3 Vind het kortste pad (i.e. het pad met de kleinste lengte) van b naar d in beide grafen.
   1.4 Vind in beide grafen de langste enkelvoudige cykel die d bevat.

HO GENT

De adjacentiematrix

We veronderstellen dat de knopen van de graaf $G = (V, E)$ genummerd zijn van 1 t.e.m. n. Wanneer we te maken hebben met een (ongewogen) graaf dan kunnen we deze voorstellen door een ADJACENTIEMATRIX A. Voor deze adjacentiematrix geldt:

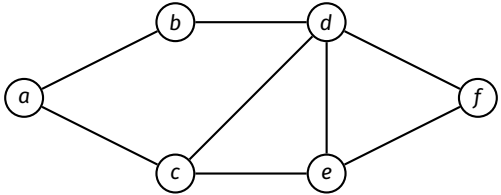
$$A_{i,j} = \begin{cases} 1 & \text{als } (i,j) \in E \\ 0 & \text{anders.} \end{cases}$$

Opmerking: Voor een ongerichte graaf is de adjacentiematrix steeds symmetrisch, i.e. A^T is steeds gelijk aan A.

HO GENT

Voorbeeld

Beschouw de graaf:

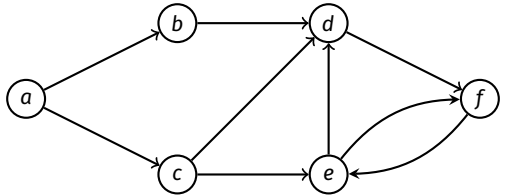


Adjacentiematrix:

$$A = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}.$$

HO GENT

Voorbeeld



Geef de adjacentiematrix van deze graaf. Wat merk je?

Adjacentiematrix kan ook gebruikt worden voor gewogen grafen.

HO GENT

Geheugen- en tijdsgebruik

- Geheugenruimte steeds $\Theta(n^2)$.
- Maximaal aantal bogen in een graaf is $\Theta(n^2)$ (waarom?)
- In een IJLE graaf wordt zo heel wat geheugen verspild.
- Bepalen of twee knopen adjacent zijn: $\Theta(1)$.
- Alle buren bepalen/overlopen: $\Theta(n)$, onafhankelijk van de graad van de knoop.

HO GENT

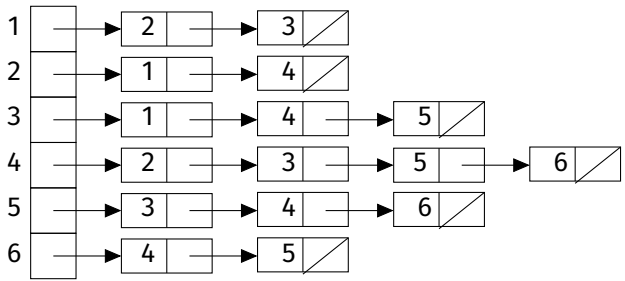
Voorbeeld gelabelde en gewogen graaf

Brugge	→ 1	∞	57	∞	∞	∞	133	∞	∞	∞	∞	∞
Gent	→ 2	57	∞	55	58	86	114	∞	∞	∞	∞	∞
Brussel	→ 3	∞	55	∞	45	30	65	∞	∞	99	35	83
Antwerpen	→ 4	∞	58	45	∞	64	∞	∞	∞	∞	∞	79
Leuven	→ 5	∞	86	30	64	∞	94	∞	∞	76	25	58
Bergen	→ 6	133	114	65	∞	94	∞	77	∞	∞	68	∞
Namen	→ 7	∞	∞	∞	∞	∞	77	∞	130	63	38	∞
Aarlen	→ 8	∞	∞	∞	∞	∞	∞	130	∞	124	∞	∞
Luik	→ 9	∞	∞	99	∞	76	∞	63	124	∞	86	55
Waver	→ 10	∞	∞	35	∞	25	68	38	∞	86	∞	∞
Hasselt	→ 11	∞	∞	83	79	58	∞	∞	∞	55	∞	∞

HO GENT

Adjacentielijst-voorstelling

De ADJACENTIELIJST-VOORSTELLING van een graaf G bestaat uit een array van toppen, genummerd 1 t.e.m. n. Op de plaats i van deze array worden, in een lineair gelinkte lijst, de buren van top i bijgehouden.
Voor de voorbeeldgraaf:



HO GENT

Geheugen- en tijdsgebruik

- De adjacentielijst-voorstelling gebruikt $\Theta(n + m)$ geheugenruimte.
- Het bepalen of twee knopen adjacent zijn kan nu *niet* langer in constante tijd gebeuren. Om te weten of i en j adjacent zijn moeten we immers de gelinkte lijst horend bij i overlopen om na te gaan of j in deze lijst aanwezig is.
- Als we alle buren van een knoop i willen overlopen dan gebeurt dit nu in een tijd die lineair is in het aantal buren van de knoop i. Dit is theoretisch de best mogelijke uitvoeringstijd.

HO
GENT

Implementatie m.b.v. “moderne” collections

- Graaf geeft relaties/verbanden tussen knopen.
- Map, of dictionary datastructuren zijn uitstekend geschikt voor het aangeven van dergelijke relaties. Bv. beeld elke knoop (of zijn label) af op de verzameling van zijn buren (bij ongewogen graaf) of op een verzameling van tupels (bij gewogen graaf).

HO GENT

Voorbeeld

Stel dat g een Map of dictionary is die de gewogen graaf met provinciehoofsteden voorstelt. Dan is

$$g[\text{"Brugge"}]$$

gelijk aan de verzameling

$$\{(\text{"Gent"}, 57), (\text{"Bergen"}, 133)\},$$

die bestaat uit twee tweetallen. Voor

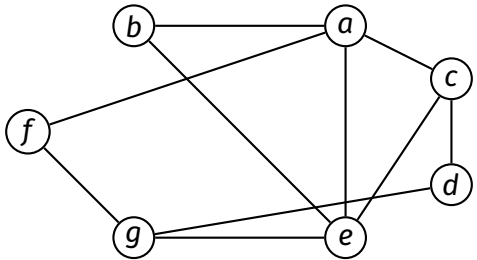
$$g[\text{"Namen"}]$$

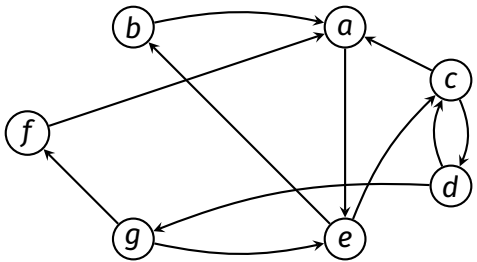
vinden we de volgende verzameling:

$$\{(\text{"Bergen"}, 77), (\text{"Aarlen"}, 130), (\text{"Luik"}, 63), (\text{"Waver"}, 38)\}.$$

HO GENT

Oefeningen





HO GENT

Oefeningen

1. Beschouw de gerichte en ongerichte graaf in Figuur 5.3 (vorige slide).
   - Geef voor beide grafen de adjacentiematrix. Je mag veronderstellen dat a rangnummer 1 heeft, b rangnummer 2 enzovoort.
   - Geef voor beide grafen de adjacentielijst-voorstelling. Je mag veronderstellen dat a rangnummer 1 heeft, b rangnummer 2 enzovoort.
2. Hoe berekent men de graad van een knoop i van een graaf G wanneer de adjacentiematrix A van de graaf gegeven is? Geef een algoritme. Wat is de tijdscomplexiteit van deze methode?
3. Hoe berekent men de graad van een knoop i van een graaf G wanneer de adjacentielijstvoorstelling van de graaf gegeven is? Geef een algoritme. Wat is de tijdscomplexiteit van deze methode?

HO GENT

Zoeken in Grafen: Inleiding

Veronderstel dat een graaf $G = (V, E)$ gegeven is. Dan kunnen we geïnteresseerd zijn om algoritmes te vinden die de volgende vragen kunnen beantwoorden:

1. Welke knopen kunnen we bereiken vanuit een knoop v?
2. Bestaat er een pad van v naar een specifieke knoop w?
3. Wat is het kortste pad van v naar w?

HO GENT

Generiek zoeken

- Doel van het algoritme: Start vanaf knoop s; vind alle knopen v waarvoor er een pad is van s naar v.
- Idee: initieel ontdekte gebied = knoop s
- Breid in elke stap het ontdekte gebied uit door een boog (u, v) te volgen die de “grens” oversteekt.
- Op deze manier voeg je v toe aan het ontdekte gebied.
- Stop wanneer je geen bogen meer kan volgen, i.e. wanneer er geen bogen meer zijn die de grens tussen ontdekt en onontdekt gebied oversteken.

HO GENT

Generiek zoeken: code

Invoer Een gerichte of ongerichte graaf $G = (V, E)$ met orde $n > 0$. Een knoop s waarvan het zoeken vertrekt. De knopen zijn genummerd van 1 tot n, i.e. $V = \{1, 2, \ldots, n\}$.

Uitvoer Een array D met $D[v]$ = true als en slechts als er een pad bestaat van s naar v.

```
1: function ZoekGeneriek(G, s)
2:     D ← [false, false, ..., false]                # n keer false
3:     D[s] ← true                                   # markeer s
4:     while ∃(u, v): D[u] = true ∧ D[v] = false do
5:         kies een boog (u, v) met D[u] = true ∧ D[v] = false
6:         D[v] ← true                               # markeer v
7:     end while
8:     return D
9: end function
```

HO GENT

Generiek Zoeken: eigenschap

Eigenschap

Wanneer het algoritme voor generiek zoeken eindigt dan geldt voor elke knoop v *van* G *dat* v *gemarkeerd is als "ontdekt" (i.e.* $D[v] = \text{true}$*) als en slechts als er een pad bestaat van* s *naar* v *in* G*.*

Bewijs.

Twee delen: zie bord! □

HO GENT

Generiek zoeken: voorbeeld

Voorbeeldgraaf uit cursus.

gemarkeerde knopen	mogelijke bogen	gekozen boog
$\{1\}$	$(1,2),(1,3)$	$(1,2)$
$\{1,2\}$	$(1,3),(2,4)$	$(1,3)$
$\{1,2,3\}$	$(2,4),(3,4),(3,5)$	$(3,4)$
$\{1,2,3,4\}$	$(3,5),(4,5),(4,6)$	$(4,6)$
$\{1,2,3,4,6\}$	$(3,5),(4,5),(5,6)$	$(3,5)$
$\{1,2,3,4,5,6\}$	geen	geen

HO GENT

Breedte-Eerst Zoeken

Idee: bezoek de knopen in “lagen”.

Eerst *s* zelf, dan de knopen die één boog verwijderd zijn van *s*, dan de knopen die twee bogen verwijderd zijn van *s*, enzovoort.

Gebruikte datastructuur: wachtrij (FIFO)

HO GENT

BFS: Pseudocode

Invoer Een gerichte of ongerichte graaf $G = (V, E)$ met orde $n > 0$. Een startknoop s; $V = \{1, 2, \ldots, n\}$.

Uitvoer Een array D met $D[v]$ = true asa $\exists$ pad van s naar v.

```
 1: function BreedteEerst(G, s)
 2:     D ← [false, false, ..., false]          # n keer false
 3:     D[s] ← true                             # markeer s
 4:     Q.init()                                # wachtrij van knopen
 5:     Q.enqueue(s)
 6:     while Q ≠ ∅ do
 7:         v ← Q.dequeue()
 8:         for all w ∈ buren(v) do
 9:             if D[w] = false then            # w nog niet ontdekt
10:                 D[w] ← true
11:                 Q.enqueue(w)
12:     return D
13: end function
```

HO GENT

Breedte-Eerst Zoeken: Voorbeeld

iteratie	Q	D
1	[1]	[T,F,F,F,F,F]
2	[2,3]	[T,T,T,F,F,F]
3	[3,4]	[T,T,T,T,F,F]
4	[4,5]	[T,T,T,T,T,F]
5	[5,6]	[T,T,T,T,T,T]
6	[6]	[T,T,T,T,T,T]
7	[]	[T,T,T,T,T,T]

HO GENT

Breedte-Eerst Zoeken: uitvoeringstijd

Eigenschap

Wanneer een adjacentielijst-voorstelling gebruikt wordt voor een graaf G, dan is de uitvoeringstijd $T(n, m)$ van Algoritme 5.2 van de grootte-orde $\Theta(n + m)$.

HO GENT

Oefening

1. Een ongerichte graaf is GECONNECTEERD als en slechts als er een pad bestaat tussen elke twee knopen *v* en *w*.
   - Ga na dat de bovenstaande definitie equivalent is met zeggen dat er een pad bestaat van een bepaalde knoop *s* naar alle andere knopen.
   - Schrijf een methode ISGECONNECTEERD die nagaat of een ongerichte graaf geconnecteerd is (return-waarde true) of niet (return-waarde false). Doe dit door de methode BREEDTEEERST aan te passen.

HO GENT

Diepte-Eerst Zoeken

Idee: zo snel mogelijk zo diep mogelijk in de graaf. (zie: http://xkcd.com/761/)

We bezoeken steeds de meest recent ontdekte knoop.

We gebruiken m.a.w. een LIFO structuur (stapel) i.p.v. een wachtrij.

We gebruiken de impliciete call-stack.

HO GENT

Diepte-Eerst Zoeken: code

Invoer Een gerichte of ongerichte graaf $G = (V, E)$ met orde $n > 0$. Een startknoop s; $V = \{1, 2, \ldots, n\}$.

Uitvoer Een array D met $D[v]$ = true asa $\exists$ pad van s naar v.

```
1:  function DiepteEerst(G, s)
2:      D ← [false, false, ..., false]                    # n keer false
3:      DiepteEerstRecursief(G, s, D)
4:      return D
5:  end function
6:  function DiepteEerstRecursief(G, v, D)
7:      D[v] ← true                                       # markeer v
8:      for all w ∈ buren(v) do
9:          if D[w] = false then                          # w nog niet ontdekt
10:             DiepteEersteRecursief(G, w, D)
11:         end if
12:     end for
13: end function
```

HO GENT

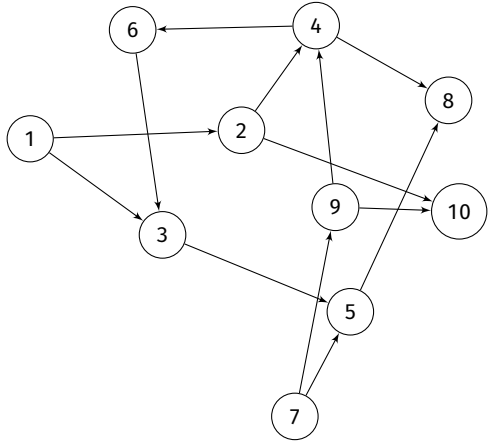
Diepte-Eerst Zoeken: Voorbeeld

DiepteEerstRecursief(*G*, 1, [false, false, false, false, false, false])
DiepteEerstRecursief(*G*, 2, [true, false, false, false, false, false])
DiepteEerstRecursief(*G*, 4, [true, true, false, false, false, false])
DiepteEerstRecursief(*G*, 3, [true, true, false, true, false, false])
DiepteEerstRecursief(*G*, 5, [true, true, true, true, false, false])
DiepteEerstRecursief(*G*, 6, [true, true, true, true, true, false])

HO GENT

Topologisch Sorteren

Precedentiegraaf van software modules.



In welke volgorde kunnen de modules gecompileerd worden?

HO GENT

Topologische Sortering: definitie

Definitie

Een TOPOLOGISCHE SORTERING van een gerichte graaf G kent aan elke knoop v een verschillend rangnummer $f(v)$ toe van 1 t.e.m. n zodanig dat de volgende eigenschap geldt:

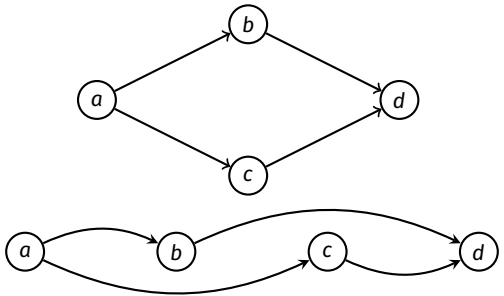
$$\forall (u, v) \in E : \; f(u) < f(v), \tag{1}$$

m.a.w. als (u, v) een boog is in de graaf dan is het rangnummer van de kop v groter dan het rangnummer van de staart u.

Wanneer we alle knopen op een rechte lijn tekenen, gesorteerd volgens het rangnummer van hun topologische sortering, dan zullen alle bogen vooruit wijzen.

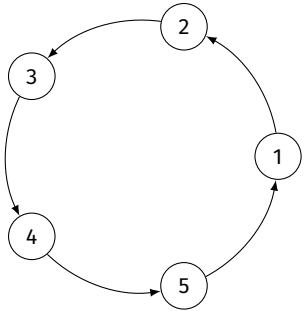
HO GENT

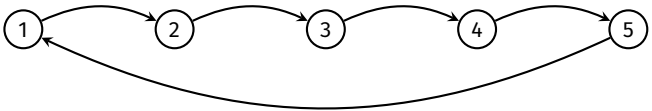
Topologische Sortering: Voorbeeld



HO GENT

Topologische Sortering: (Mislukt) Voorbeeld





HO GENT

Eigenschap Topologisch Sorteren

Eigenschap

Wanneer een gerichte graaf G geen enkelvoudige cykels heeft, dan bestaat er een topologische sortering van G.

Bewijs.

- Een gerichte graaf G zonder cykels heeft een knoop zonder buren.
- De knoop zonder buren is een goede kandidaat om rangnummer n te krijgen.
- Eenvoudig algoritme om topologische sortering te vinden.

HO GENT

Topologisch Sorteren: Algoritme

We kunnen *diepte-eerst zoeken* aanpassen om een topologische sortering te geven.

Idee: DFS zoekt vanuit elke knoop v alle knopen w waarvoor er een pad van v naar w bestaat. In een topologische sortering moet v dus vóór w komen.

Hou een lijst S bij; wanneer v is 'afgewerkt' met DFS, voeg dan v vooraan toe.

DFS moet eventueel meerdere malen worden opgeroepen!

HO GENT

Topologisch Sorteren: Ontdekken Cykel

- Initieel alle knopen op 0 (*onontdekt*)
- Volledig afgewerkte knopen op 2 (*afgewerkt*) (deze knopen hebben al een plaats in de sortering)
- Knopen waarvoor we nog zoeken naar opvolgers op 1 (*bezig*)

Stel we zijn bezig met de buren van knoop v (dus v op 1). We vinden buur w met toestand 'bezig'. Dit betekent dat er een pad is van w naar v; samen met de boog (v, w) wordt dit een cykel!

HO GENT

Topologisch Sorteren: code

Invoer Een gerichte graaf $G = (V, E)$ met orde $n > 0$. De knopen zijn genummerd van 1 tot n, i.e. $V = \{1, 2, \ldots, n\}$.

Uitvoer Een topologische sortering van G indien mogelijk, false anders.

```
1:  function SorteerTopologisch(G)
2:      global cycleDetected ← false                    # globale variabele
3:      D ← [0, 0, ..., 0]                              # n keer 0
4:      S ← ∅                                           # S is lege lijst
5:      for all s ∈ V do
6:          if D[s] = 0 then                            # s nog niet gezien
7:              DfsTopo(G, s, D, S)                     # S en D referentieparameters
8:              if cycleDetected = true then            # controleer op cykel
9:                  return false
10:             end if
11:         end if
12:     end for
13:     return S
```

HO GENT

Code: vervolg

```
1:  function DfsTopo(G, v, D, S)
2:      D[v] ← 1                                  # markeer v als 'bezig'
3:      for all w ∈ buren(v) do
4:          if D[w] = 0 ∧ cycleDetected = false then   # w nog niet ontdekt
5:              DfsTopo(G, w, D, S)
6:          else if D[w] = 1 then                 # cykel ontdekt w ⇝ v → w
7:              cycleDetected ← true
8:          end if
9:      end for
10:     D[v] ← 2                                  # markeer v als 'voltooid'
11:     voeg v vooraan toe aan S                  # ken rangnummer toe aan v
12: end function
```

HO GENT

Voorbeeld: softwaremodules

Uitwerking op het bord.

HO
GENT

Oefeningen

2. Vind alle mogelijke topologische sorteringen van de graaf in Figuur 5.10.
3. Vind de compilatievolgorde van de modules in Figuur 5.9 wanneer de labels in dalende volgorde worden doorlopen.
4. Veronderstel nu dat er in de graaf van Figuur 5.9 een extra boog $(8, 6)$ wordt toegevoegd. Pas nu het algoritme voor topologisch sorteren toe.

HO GENT

Kortste Pad in een Ongewogen Graaf

We wensen te weten hoeveel stappen (bogen) men nodig heeft om, startend vanaf een top s, een andere top v te bereiken.

We zoeken m.a.w. de lengte van een kortste pad.

Breedte-Eerst Zoeken overloopt de graaf 'laag per laag'. Kleine aanpassing nodig om kortste pad bij te houden.

HO GENT

Pseudocode

Invoer Een gerichte of ongerichte ongewogen graaf $G = (V, E)$. Een startknoop s; $V = \{1, 2, \ldots, n\}$.

Uitvoer De array D met $D[v]$ de kortste afstand van s tot v; als $D[v] = \infty$ dan is er geen pad van s naar v.

```
 1: function KortstePadOngewogen(G, s)
 2:     D ← [∞, ∞, ..., ∞]                                  # n keer ∞
 3:     D[s] ← 0                          # kortste pad van s naar zichzelf heeft lengte 0
 4:     Q.init()                                            # wachtrij van knopen
 5:     Q.enqueue(s)
 6:     while Q ≠ ∅ do
 7:         v ← Q.dequeue()
 8:         for all w ∈ buren(v) do
 9:             if D[w] = ∞ then                            # w nog niet ontdekt
10:                 D[w] ← D[v] + 1
11:                 Q.enqueue(w)
12:     return D
13: end function
```

HO GENT

Kortste Pad in een Ongewogen Graaf: Voorbeeld

	1	2	3	4	5	6
(a)	0	∞	∞	∞	∞	∞
(b)	0	1	1	∞	∞	∞
(c)	0	1	1	2	∞	∞
(d)	0	1	1	2	2	∞
(e)	0	1	1	2	2	3

HO GENT

Kortste Pad in Gewogen Graaf

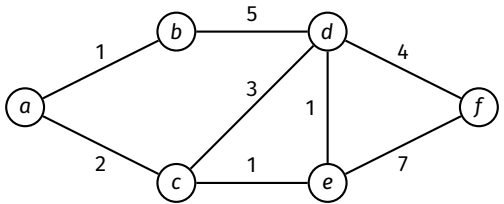
In een *gewogen* graaf willen we meestal pad met kleinste gewicht (en niet noodzakelijk met het minst aantal bogen).
Naïeve aanpassing van breedte-eerst:

$$D[w] \leftarrow D[v] + \text{gewicht}(v, w).$$

(i.p.v. $D[w] \leftarrow D[v] + 1$)

HO GENT

Toepassing Naïeve Aanpassing



Wat wordt de afstandenarray?

HO GENT

Algoritme van Dijkstra

Sleutelideeën:

1. Op elk moment: een verzameling S van knopen v waarvoor de kortste afstand van s tot v reeds gekend is, en een verzameling Q van knopen waarvoor de kortste afstand nog *niet* met zekerheid gekend is.
2. Voor elke knoop v (die tot Q behoort): $D[v]$ is de kortste afstand van een pad van s naar v *dat enkel uit knopen van S bestaat* (behalve de laatste).
3. We voegen telkens dié knoop v van Q toe aan S waarvoor $D[v]$ minimaal is onder alle knopen van Q. Dit betekent dat voor de buren w van v die tot Q behoren we eventueel $D[w]$ moeten aanpassen. Het pad van s naar v (dat nu enkel uit knopen van S bestaat) uitgebreid met de boog (v, w) zou eventueel korter kunnen zijn dan het tot dan toe gevonden kortste pad.

HO GENT

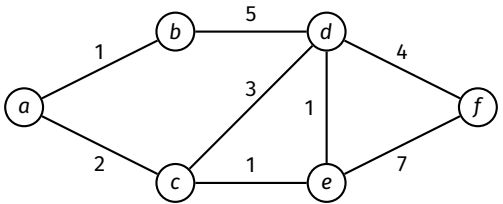
Dijkstra: Pseudocode

Invoer Een gewogen graaf $G = (V, E)$ met positieve gewichten. Startknoop s; $V = \{1, 2, \ldots, n\}$.

Uitvoer De array D met $D[v]$ de kortste afstand van s tot v; als $D[v] = \infty$ dan is er geen pad van s naar v.

```
1:  function DIJKSTRA(G, s)
2:      D ← [∞, ∞, ..., ∞]                                   # n keer ∞
3:      D[s] ← 0                          # kortste pad van s naar zichzelf heeft lengte 0
4:      Q ← V                    # knopen waarvan kortste afstand nog niet is bepaald
5:      while Q ≠ ∅ do
6:          zoek v ∈ Q waarvoor D[v] minimaal is (voor knopen in Q)
7:          verwijder v uit Q
8:          for all w ∈ buren(v) ∩ Q do
9:              if D[w] > D[v] + gewicht(v, w) then
10:                 D[w] ← D[v] + gewicht(v, w)                # korter pad via v → w
11:     return D
12: end function
```

Dijkstra: Voorbeeld



Pas Dijkstra toe met startknoop a.

HO GENT

Implementatie

In Dijkstra moeten herhaaldelijk minima worden berekend: prioriteitswachtrij (bv. binaire hoop)!

Probleem: sleutels moeten worden aangepast (verminderd).

Standaard niet mogelijk met een binaire hoop. Uitbreiding mogelijk m.b.v. tweede array.

HO GENT

Oefeningen

1. In Algoritme 5.5 wordt nu enkel de afstand van elke knoop v tot de startknoop s bijgehouden. In veel toepassingen heeft men echter ook een pad nodig dat deze minimale afstand realiseert.
   - Pas de pseudo-code van Algoritme 5.5 aan zodanig dat er een tweede array P wordt teruggeven zodanig dat $P[v]$ de knoop geeft die de voorganger is van v op een kortste pad van s naar v.
   - Pas je aangepaste algoritme toe op de ongerichte graaf in Figuur 5.9 startend vanaf knoop 1. Ga ervan uit dat knopen steeds worden bezocht in stijgende volgorde. Hoe zit de array P er uit na afloop?
   - Schrijf een algoritme dat als invoer de array P neemt en een knoop v. Het algoritme geeft een lijst terug die het kortste pad van s naar v bevat (in de juiste volgorde).

HO GENT

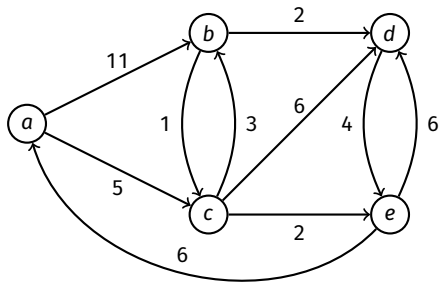
Oefeningen

2. Beschrijf hoe je volgend probleem kan oplossen als een kortste pad probleem. Gegeven een lijst van Engelstalige 5-letterwoorden. Woorden worden *getransformeerd* door juist één letter van het woord te vervangen door een andere letter. Geef een algoritme dat nagaat of een woord w_1 omgezet kan worden in een woord w_2. Indien dit het geval is dan moet je algoritme ook de tussenliggende woorden tonen voor de kortste sequentie van transformaties die w_1 in w_2 omzet.
3. Vind voor de graaf in Figuur 5.4 de lengte van het kortste pad van Brugge naar alle andere steden. Voer hiertoe het algoritme van Dijkstra uit.

HO GENT

Oefeningen

4. Vind voor de graaf in Figuur 5.14 (de lengte van) het kortste pad van de knoop a naar alle andere knopen. Voer hiertoe het algoritme van Dijkstra uit (en houd ook bij wat de kortste paden zijn).



HO GENT

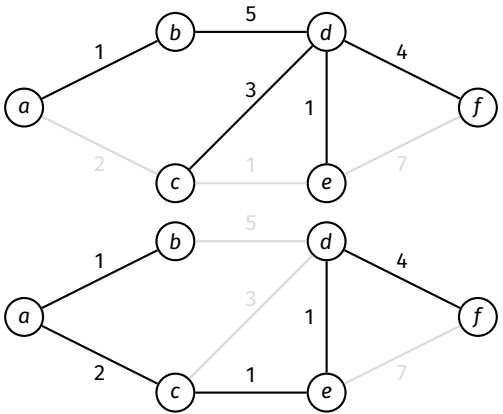
Opspannende Boom: Definitie

Definitie

Een OPSPANNENDE BOOM van een ongerichte graaf $G = (V, E)$ is een verzameling van bogen T, met $T \subseteq E$, zodanig dat $G' = (V, T)$ een pad heeft tussen elke twee knopen van V, en zodanig dat G' geen enkelvoudige cykels heeft.

HO GENT

Opspannende Bomen: Voorbeeld



HO GENT

Minimale Kost Opspannende Boom

Het gewicht van een boom is de som van de gewichten van zijn bogen:

$$\text{gewicht}(T) = \sum_{t \in T} \text{gewicht}(t).$$

Definitie

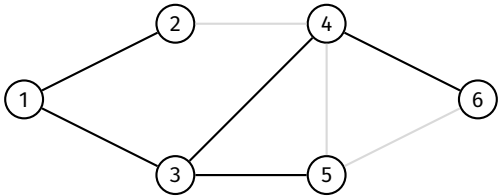
Een MINIMALE KOST OPSPANNENDE BOOM T van de graaf G is een opspannende boom zodanig dat voor alle (andere) opspannende bomen T' van G geldt dat

$$\sum_{t \in T} \text{gewicht}(t) \leq \sum_{t' \in T'} \text{gewicht}(t').$$

HO GENT

Algoritme van Prim

Het algoritme voor generiek zoeken levert reeds een opspannende boom! (Als we de bogen zouden bijhouden).
Dit is bijvoorbeeld een mogelijke uitvoer van generiek zoeken:



Essentie Prim: doe generiek zoeken maar kies steeds de *goedkoopste* beschikbare boog: een *gulzige* strategie.

HO GENT

Algoritme van Prim: Code

Invoer Een ongerichte gewogen graaf $G = (V, E)$ met orde $n > 0$. De knopen zijn genummerd van 1 tot n, i.e. $V = \{1, 2, \ldots, n\}$.

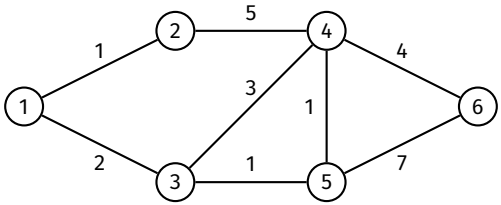
Uitvoer Een verzameling T van bogen die een minimale kost opspannende boom is.

```
 1: function PRIM(G)
 2:     D ← [false, false, ..., false]                    # n keer false
 3:     D[1] ← true                                       # kies knoop 1 als startknoop
 4:     T ← ∅                                             # gekozen bogen
 5:     while ∃(u, v): D[u] = true ∧ D[v] = false do
 6:         kies (u, v) met D[u] = true ∧ D[v] = false met minimaal gewicht
 7:         D[v] ← true
 8:         T ← T ∪ {(u, v)}                              # Voeg boog (u, v) toe aan boom
 9:     end while
10:     return T
11: end function
```

HO GENT

Algoritme van Prim: Voorbeeld

Voer het algoritme van Prim uit voor de volgende voorbeeldgraaf.



HO GENT

Algoritme van Kruskal

Alternatief algoritme om minimale kost opspannende boom te bepalen.

Eveneens een gulzig algoritme. We kiezen steeds de goedkoopste boog.

Bogen zijn niet steeds met elkaar verbonden.

Boog wordt enkel gekozen indien die *geen cykel* veroorzaakt.

HO
GENT

Algoritme van Kruskal: Code

Invoer Een ongerichte gewogen graaf $G = (V, E)$ met orde $n > 0$. De knopen zijn genummerd van 1 tot n, i.e. $V = \{1, 2, \ldots, n\}$.

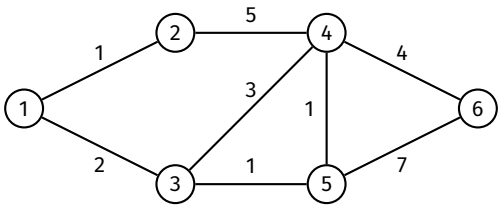
Uitvoer Een verzameling T van bogen die een minimale kost opspannende boom is.

```
1:  function KRUSKAL(G)
2:      T ← ∅                                   # Start met lege boom
3:      E' ← sorteer E volgens stijgend gewicht
4:      for all e' ∈ E' do
5:          if T ∪ e' heeft geen cykel then
6:              T ← T ∪ e'
7:          end if
8:      end for
9:      return T
10: end function
```

HO GENT

Algoritme van Kruskal: Voorbeeld

Voer het algoritme van Kruskal uit voor de volgende voorbeeldgraaf.



HO GENT

Oefeningen

1. Vind een minimale kost opspannende boom m.b.v. het algoritme van Prim voor de graaf in Figuur 5.4. Neem als startknoop “Brugge”.
2. Vind een minimale kost opspannende boom m.b.v. het algoritme van Kruskal voor de graaf in Figuur 5.4.

HO GENT

Het Handelsreizigersprobleem

Definitie

Het HANDELSREIZIGERSPROBLEEM is het volgende: gegeven een complete[1] gewogen ongerichte graaf G met niet-negatieve gewichten, vind dan een ordening van de knopen zodanig dat elke knoop juist éénmaal wordt bezocht (behalve de start- en eindknoop die samenvallen) en zodanig dat de som van de gewichten van de gekozen bogen minimaal is.

HO GENT

[1] Een graaf is compleet als er een boog is tussen elke twee (verschillende) knopen.

Moeilijkheidsgraad

Voor alle vorige problemen hebben we steeds een efficiënt (polynomiaal) algoritme kunnen geven.

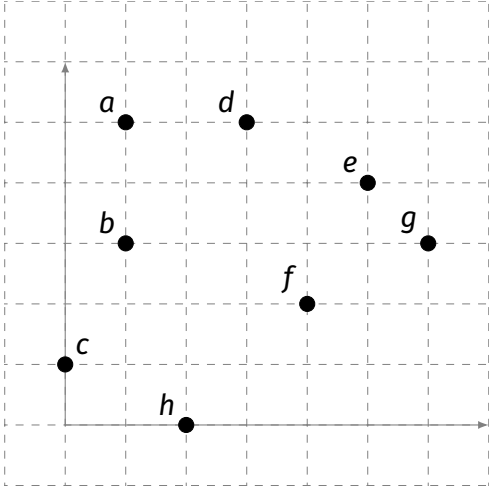
Het handelsreizigersprobleem is echter een NP-COMPLEET probleem, wat hoogstwaarschijnlijk betekent dat er *geen* polynomiaal algoritme bestaat dat alle gevallen correct kan oplossen.

Triviaal algoritme: probeer alle mogelijkheden en selecteer de beste. Aantal mogelijkheden is echter $(n-1)!$ (Je kan de eerste stad steeds als 'vast' beschouwen.)

Men gebruikt dan ook vaak *benaderende* algoritmen.

HO GENT

Steden op een grid



HO GENT

Driehoeksongelijkheid

Voor de vorige graaf is het steeds korter om rechtstreeks van v naar w te gaan dan om een omweg te maken via u. De graaf voldoet aan de driehoeksongelijkheid.

Definitie

Een graaf G voldoet aan de DRIEHOEKSONGELIJKHEID wanneer voor alle knopen u, v en w geldt dat

$$\text{gewicht}(v, w) \leq \text{gewicht}(v, u) + \text{gewicht}(u, w).$$

HO GENT

Benaderingsalgoritme voor Handelsreizigersprobleem

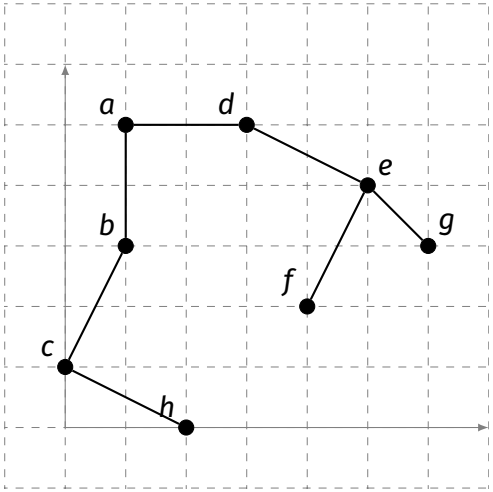
1. Bereken een minimale kost opspannende boom T voor de graaf.
2. Kies willekeurig een wortel r van deze boom.
3. Geef de cykel terug die correspondeert met het in *preorde* doorlopen van deze boom.

Wanneer de graaf G aan de driehoeksongelijkheid voldoet dan is de gevonden oplossing hoogstens tweemaal zolang als de optimale oplossing.

HO GENT

Voorbeeld: Stap 1

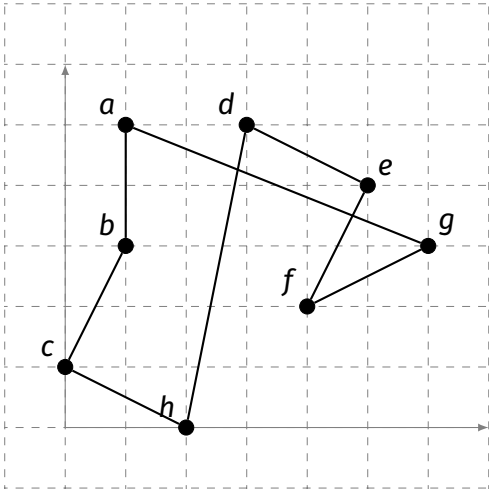
Stap 1: Minimale kost opspannende boom.



HO GENT

Voorbeeld: Stap 2 en 3

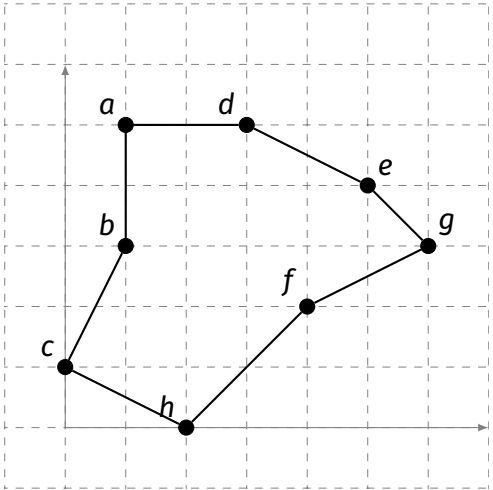
Stap 2 en 3: Wortel *a*: preorde doorlopen



HO GENT

Voorbeeld

Vergelijken met optimale oplossing:



HO GENT

Oefening

1. Beschouw opnieuw de acht steden in Figuur 5.18, maar veronderstel nu dat het gewicht van een boog gegeven wordt door de zogenaamde Manhattan-distance tussen de twee knopen, dus

$$d\big((x_1, y_1), (x_2, y_2)\big) = |x_2 - x_1| + |y_2 - y_1|$$

   1.1 Ga na dat de Manhattan-distance aan de driehoeksongelijkheid voldoet. (**Hint**: voor de absolute waarde geldt dat $|x + y| \leq |x| + |y|$.)

   1.2 Pas het benaderende algoritme voor het oplossen van het handelsreizigersprobleem toe op dit probleem. Gebruik Kruskals algoritme om de minimale opspannende boom te construeren. Wanneer meerdere bogen kunnen gekozen worden, kies dan steeds de lexicografisch kleinste boog. Neem de knoop a als wortel van de opspannende boom.

HO GENT