Inhoud

[5.1 Terminologie m.b.t grafen 2](#_Toc91447846)

[5.2 Datastructuren voor grafen 3](#_Toc91447847)

[5.2.1 De adjacentiematrix 3](#_Toc91447848)

[Geheugen- en tijdsgebruik 3](#_Toc91447849)

[5.3 Zoeken in grafen 5](#_Toc91447850)

[5.3.1 Generiek zoeken 5](#_Toc91447851)

[5.3.2 Breedte-Eerst zoeken 7](#_Toc91447852)

[5.3.3 Diepte-Eest zoeken 8](#_Toc91447853)

[5.3.4 Toepassing: Topologisch sorteren 9](#_Toc91447854)

[5.4 Kortste Pad algoritmen 12](#_Toc91447855)

[5.4.1 Kortste Pad in een Ongewogen Graaf 12](#_Toc91447856)

[5.4.2 Dijkstra’s Algoritme 12](#_Toc91447857)

[Implementatiedetails 14](#_Toc91447858)

[5.5 Minimale Kost Opspannende Bomen 15](#_Toc91447859)

[5.5.1 Minimale Kost Opspannende Bomen 15](#_Toc91447860)

[5.5.2 Prims Algoritme 16](#_Toc91447861)

[5.5.3 Kruskals algoritme 17](#_Toc91447862)

Hoofdstuk 5 Graafalgoritmes

# 5.1 Terminologie m.b.t grafen

In heel wat situaties heeft men te maken met objecten die verbonden zijn door een bepaalde relatie: steden zijn met elkaar verbonden via wegen, computers met netwerkkabels,… in al deze situaties is het vaak zinvol om een bepaalde **kost**  te associëren met een verbinding, bv de afstand in km, de communicatiesnelheid,…

Al deze situaties kunnen gemodelleerd worden adhv een **graaf.**

**Definitie:** een **graaf** **G** bestaat uit een verzameling **knopen** **V** en een **verzameling bogen** **E**. Elke boog verbind twee knopen. We noteren dus e=(v, w)

De graaf G wordt genoteerd als het koppel (V,E), => G = (V,E).

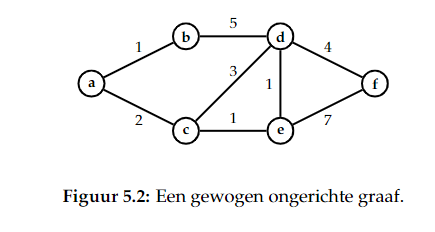
Wanneer v en w verbonden zijn, zijn ze adjacent. We zeggen dan ook dat ze incident zijn met e, en omgekeerd dat e incident is met v en w. **De buren** van een knoop v zijn alle knopen w die adjacent zijn met v. Het aantal buren van een knoop v noemt met de **graad** van deze knoop.

Afbeelding met horloge

Automatisch gegenereerde beschrijving

In de gerichte graaf is de graad van top c 2

In de meeste gevallen is het handig extra informatie op te nemen in de bogen, in dit geval spreekt men van een gewogen graaf. Deze informatie kan bv de afstand zijn tussen 2 steden in Km.



**Definitie:** een pad in een **graaf** g is een opsomming van toppen (v1, v2, v3,…) zodanig dat er een boog bestaat tussen vi en vi+1 voor i ∈ {1,2,….,k-1}. **De lengte** van dit pad is k-1, zijnde het aantal bogen op dit pad.

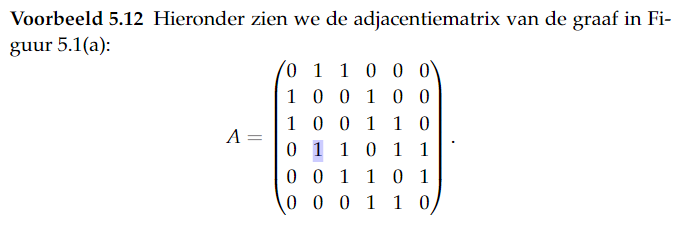
**Definitie:** Een enkelvoudige cykel in een graaf is een pad waarvan de lengte strikt positief is en dat begint en eindigt in dezelfde knoop, waarbij alle knopen (behalve start en eind).

# 5.2 Datastructuren voor grafen

## 5.2.1 De adjacentiematrix

Afbeelding met tekst

Automatisch gegenereerde beschrijvingWe 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 matrix geldt:

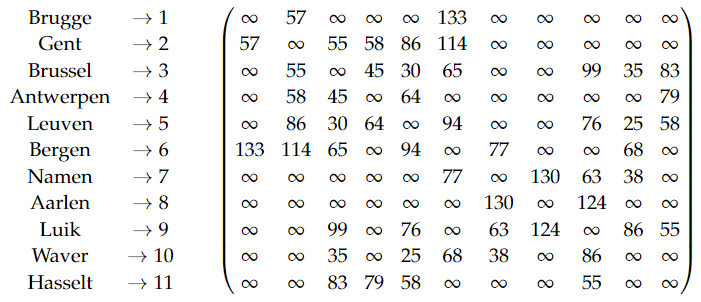


Aangezien de graaf ongericht is, hebben we inderdaad dat A een symmetrische matrix is: A^t = A.

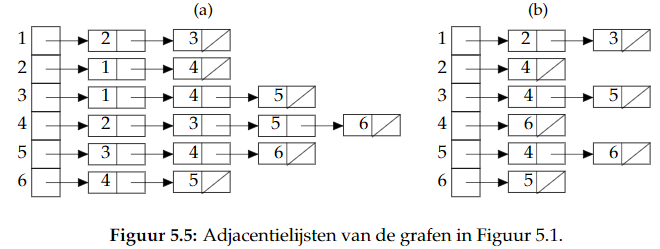
### Geheugen- en tijdsgebruik

De **geheugenruimte** die een adjacentiematrix in beslag neemt is steeds 𝛩(𝑛2). Wanneer het aantal bogen klein is in vergelijking met het maximale aantal bogen (n² in een gerichte graaf), dab verspilt deze voorstellingswijze heel wat geheugenruimte. => dit noemt men een IJLE graaf.

Het bepalen of twee knopen adjacent zijn of niet gebeurt in constante tijd 𝛩(1).



Merk op dat we als speciale waarde hier ∞ gebruikt hebben. We duiden aan dat de graaf geen lus heeft door op die plaatsen ∞ te zetten.



De adjacentielijst-voorstelling van graaf G bestaat uit een array van toppen, genummerd 1 tem n. Op de plaats i van deze array worden in een lineair gelinkte lijst, de buren van top i bijgehouden.

* De adjacentielijst-voorstelling gebruikt 𝛩(𝑛 + 𝑚) geheugen ruimte.
* 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.

# 5.3 Zoeken in grafen

Veronderstel dat een graaf G = (V,E) gegeven is. We zijn geïnteresseerd om algoritmes te vinden die volgende zaken beantwoorden.

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

## 5.3.1 Generiek zoeken

Het doel van een zoek algoritme bestaat erin om in graaf G vanaf een gegeven knoop v alle knopen te vinden die bereikbaar zijn vanaf v.

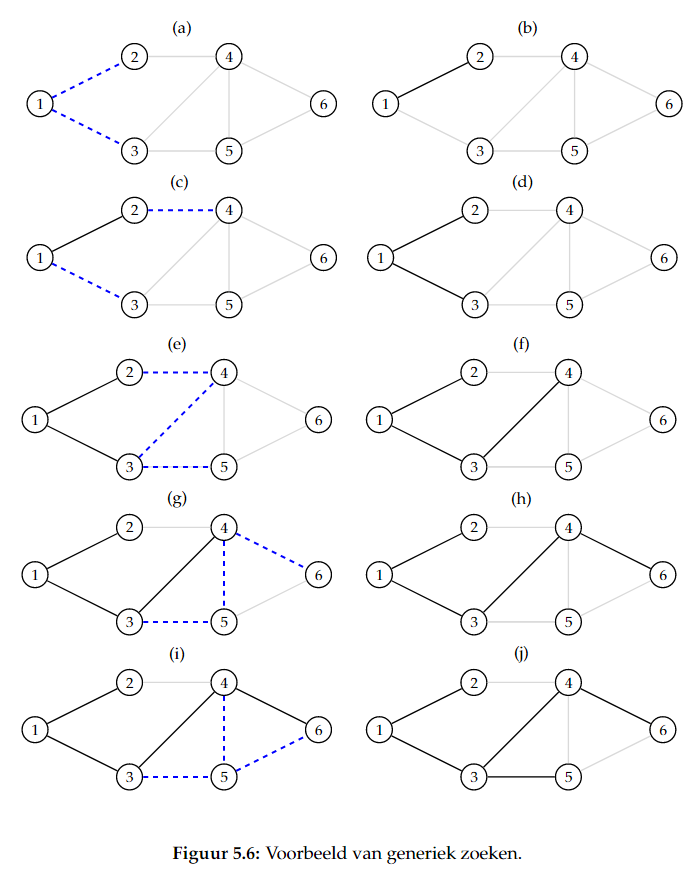
We markeren de knoop s als ontdekt. In elke stap breiden we het ontdekte gebied uit door een boog (u,v) waarbij u reeds ontdekt is en v nog niet ontdekt is. Op deze manier breiden we het ontdekte gebied uit met knoop v.

**Pseudocode:**

Afbeelding met tekst

Automatisch gegenereerde beschrijving

**Eigenschap:** wanneer het algoritme voor generiek zoeken eindigt, dan geldt voor elke knoop v van G dat v gemarkeerd is als ‘ontdekt’



We sommen het verloop van het algoritme op:

Afbeelding met tafel

Automatisch gegenereerde beschrijving

Merk op dat dit algoritme de bogen niet op een systematische manier afloopt.

## 5.3.2 Breedte-Eerst zoeken

Dit algoritme is een instantie van het algoritme voor generiek zoeken. Hier gaan we de knopen van een graaf G bezoeken in ‘lagen’. Eerst bezoeken we de knoop s zelf, daarna de knopen die juist een boog verwijderd zijn.

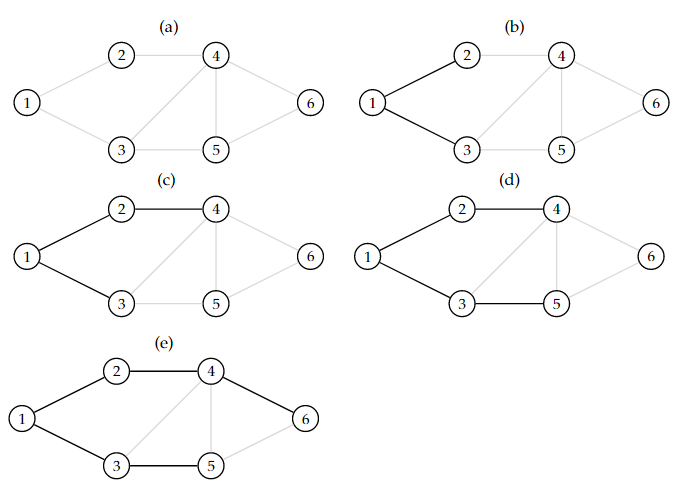
Deze datastructuur implementeert de FIFO-datastructuur, i.e een wachtrij. Deze wachtrij zal de knopen v bevatten die reeds zijn ontdekt, maar waarvoor er mogelijks nog geen buren u zijn ontdekt.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Afbeelding met tafel

Automatisch gegenereerde beschrijving

Het is gemakkelijk om te controleren dat de wachtrij op elk moment knopen bevat van hoogstens twee verschillende ‘lagen’

.

1. Het initialiseren van de array D neemt tijd O(n).
2. Elke knoop wordt hoogstens een keer toegevoegd aan de wachtrij, en wordt er dus hoogstens 1x uit verwijderd. De totale tijd voor de lus op regel 8 is dus hoogstens.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

We vinden dus dat de uitvoeringstijd O(n+m) is aangezien alle bewerkingen in de lus een constante uitvoeringstijd hebben.

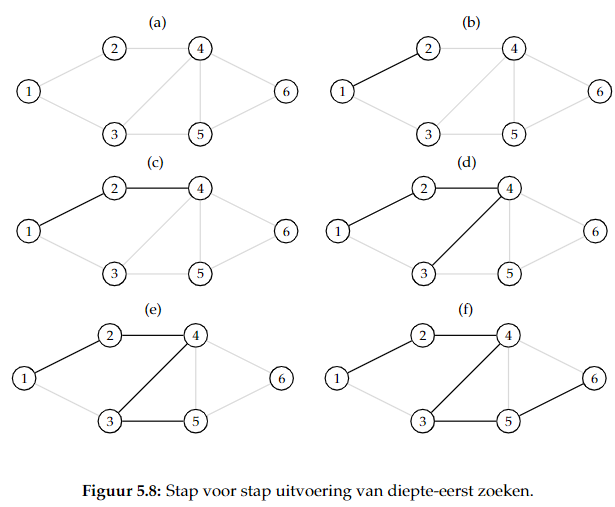
## 5.3.3 Diepte-Eerst zoeken

Een instantie van het generieke zoekalgortime. Bij diepte-eerst zoeken gaan we zo snel mogelijk zo diep mogelijk in de graaf: we bekijken steeds de buren van de ‘diepste’ ontdekte knoop.

Dit algoritme is gelijkaardig aan breedte-eerst zoeken, maar we vervangen de wachtrij van breedte-eerst zoeken door een stapel (stack).

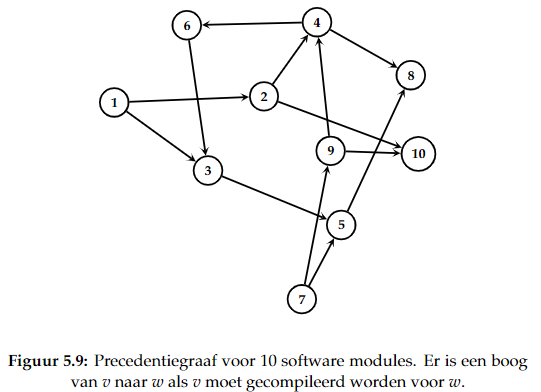
Afbeelding met tekst

Automatisch gegenereerde beschrijving



Merk op dat het eerste statement van DiepteEesrstRecursief ook aan D[6] de waarde true zal toekennen.

## 5.3.4 Toepassing: Topologisch sorteren



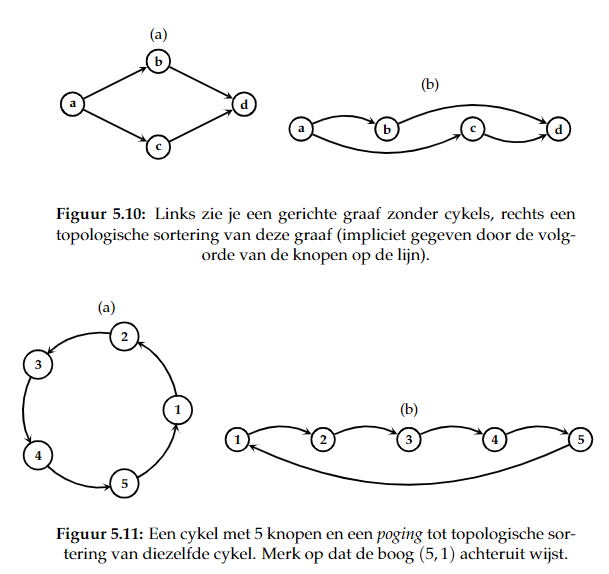
We veronderstellen dat G een gerichte graaf is, zonder enkelvoudige cykels. Een voorbeeld hiervan is een **precendentiegraaf**, waarbij de verschillende knopen taken voorstellen.

* Er is een boog v naar w als taak v moet afgewerkt zijn alvorens taak w kan aangevat worden.

Een topologische sortering is een volgorde van de taken zodanig dat voor elke taak w al zijn voorgaande taken zijn afgewerkt alvorens w wordt gestart.

**Def:** 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:

* Als (u,v) een boog is in de graaf, dan is het rangnummer van de kop v kleiner 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 vooruitwijzen.



**Eigenschap topologisch sorteren:** 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

= > veronderstel dat G een gerichte graaf is zonder cykles maar dat elke knoop minstens een buur heeft. We kunnen dan een pad (v1,v2,v3,v4,v5,vn+1) vormen.

Het pad bestaat uit n+1 knopen, maar de graaf G heeft slechts n knopen. Dit betekent dat een knoop herhaald word in dit pad en zo krijgen we een cykel G. dit is in strijd met de veronderstellen dat G geen cykel bevat. Er moet dus een knoop zijn zonder buren.

Het is duidelijk dat wanneer G een topologische sortering heeft, dat dan de knoop met rangnummer n een knoop is zonder buren. Veronderstel dat v een knoop is die wel buren heeft en dat f(v) = n, dan betekent dit volgens eigenschap (5.1) dat f(u) > n wanneer (v,u) ∈ E, maar dit is onmogelijk aangezien het grootste rangnummer dat we kunnen toe kennen nis. M.a.w. in een topologische sortering zijn de enige knopen v waarvoor f(v) = n kan zijn die knopen waarvoor buren(v) = ∅.

Een eenvoudig algoritme om een topologische sortering te vinden is dan het volgende.

1. Kies een knoop v zonder buren (zo is er minstens een) en stel f(v) = n
2. Doe nu (recursief) hetzelfde voor de graaf G –{v}²

Dit algoritme werkt om de volgende reden: veronderstel dat de knoop v het rangnummer i krijgt, dan heeft deze knoop in de huidige graaf geen buren. Alle bogen die vertrekken vanuit v zijn dus reeds verwijderd uit de graaf G. Dat betekent dat voor een boog (v, u) er steeds geldt dat f(u) > i.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Afbeelding met tafel

Automatisch gegenereerde beschrijving

# 5.4 Kortste Pad algoritmen

## 5.4.1 Kortste Pad in een Ongewogen Graaf

Wanner een graaf G ongewogen is, dan is een kortste pad van knoop v naar knoop u een pad van v naar u dat het kleinste aantal bogen bevat. Omdat bij breedte-eerst zoeken de graaf laag per laag overlopen wordt kunnen we het algoritme gemakkelijk aanpassen om voor elke knoop v de lengte van het kortste pad van s naar v terug te geven.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

## 5.4.2 Dijkstra’s Algoritme

Wanneer de graaf gewogen is (wanneer elke boog e een gewicht heeft) dan is men meestal niet geïnteresseerd in het pad met het minste bogen, maar in het pad met het kleinste totale gewicht. In dat geval zal algoritme 5.5 niet langer correct zijn.

Een algoritme dat in het geval alle gewichten positief zijn het juiste antwoord geeft, is het **Dijkstra’s** algoritme.

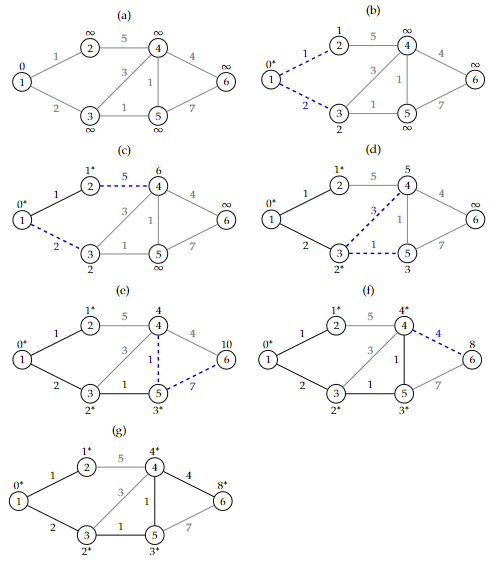
1. Op elk moment in het algoritme zijn de knopen verdeeld in twee disjuncte verzamelingen:

* Een verzameling S van knopen v waarvoor de kortste afstand van s tot v al gekend is
* Een verzameling Q van knopen waarvoor de kortste afstand nog niet met zekerheid gekend is.

1. Voor elke knoop v die tot Q behoort houden we steeds de korste afstand D[v] bij van een pad van s naar v.
2. We voegen telkens die knoop v van Q toe aan S waarvoor D[v] minimaal is onder alle knopen van Q. voor de buren w van v die tot Q behoren moeten we eventueel D[w] aanpassen. Het pad van s naar v uitgebreid met de boog (v,w) zou eventueel korter kunnen zijn dan het tot dan toe gevonden kortste pad.

Afbeelding met tekst

Automatisch gegenereerde beschrijving



### Implementatiedetails

We zien in het Dijkstra algoritme dat er herhaalde minimum berekeningen gebeuren. Op regel 6 moet herhaaldelijk de knoop v gevonden worden met de minimale waarde voor D[v]. Een binaire hoop is hier zeer geschikt voor aangezien deze geen prioriteitswachtrij implementeert.

Een efficiënte implementatie zal dus gebruik maken van een binaire hoop waarbij steeds aan volgende zaken voldaan is:

1. De elementen in de hoop zijn de elementen van Q
2. De sleutel voor elk element v is de waarde van D[v]

Op regel 10 moet echter de sleutelwaarde van w verkleind worden. Wanneer men zou weten wat de positie is van w in de binaire hoop, dan ken men gemakkelijk de sleutelwaarde aanpassen.

* Dit komt neer op het laten omhoog bubbelen van w tot op zijn juiste plaats.

Men houdt dus een array P bij waarin voor elke knoop v zijn positie in een binaire hoop wordt bijgehouden.

# 5.5 Minimale Kost Opspannende Bomen

## 5.5.1 Minimale Kost Opspannende Bomen

Veronderstel dat een wegennetwerk gegeven is dat ervoor zorgt dat we van elke stad naar elke andere stad kunnen rijden, dan kunnen we ons afvragen welke wegen essentieel zijn om van elke stad naar elke andere stad te kunnen rijden.

* Wanneer het aantal wegen minimaal is, dan spreken we van een opspannende boom.

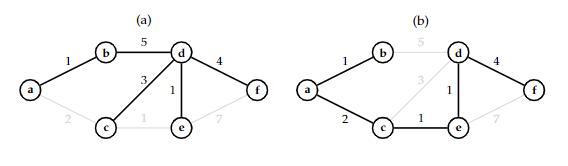
**Definitie:** een opspannende boom van een ongerichte graaf G = (V,E) is een verzameling van bogen T, met T ⊆ E, zodanig dat G′= (V,T) een pad heeft tussen elke twee knopen van V, en zodanig dat G’ geen enkelvoudige cykels heeft.

In het algemeen heeft een ongericht graaf veel opspannende bomen. We zijn geïnteresseerd in de opspannende boom of bomen met het minimale gewicht, waarbij het gewicht van de boom gedefinieerd wordt als de som van de gewichten van zijn bogen



**Definitie:** een minimale kost opspannende boom T van de graaf G is een opspannende boom zodanig dat voor alle andere opspannende bogen T’ van G geldt dat:





Men ziet twee opspannende bomen voor dezelfde graaf. We controleren of deze opspannend zijn:

1. Het is duidelijk dat ze geen cykel bevatten
2. Men ziet eenvoudig dat er een pad is tussen elke twee knopen van de graaf.

Voor figuur a bekomen we een totaal gewicht van 14.

Voor figuur b bekomen we een totaal gewicht van 9.

## 5.5.2 Prims Algoritme

Het algoritme voor generiek zoeken kan eenvoudig aangepast worden om een opspannende boom terug te geven. Het algoritme voor generiek zoeken genereert blijkbaar als een opspannende boom. We dienen enkele nog bij te houden welke bogen gekozen worden.

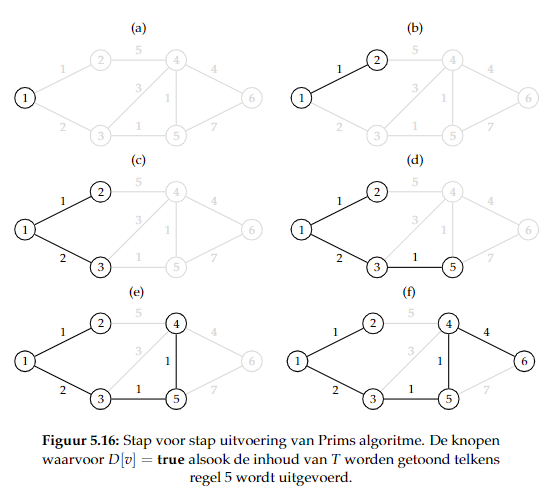
Afbeelding met tekst

Automatisch gegenereerde beschrijving

We voeren Prims algoritme uit op de gewogen graaf, we zien in de tabel in welke volgorde de bogen gekozen worden:

Afbeelding met tafel

Automatisch gegenereerde beschrijving



## 5.5.3 Kruskals algoritme

Net als Prims algoritme is het een gulzig algoritme, maar daar waar Prims algoritme de gekozen bogen steeds met elkaar verbonden zijn is dat in Kruskals algoritme niet het geval.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

