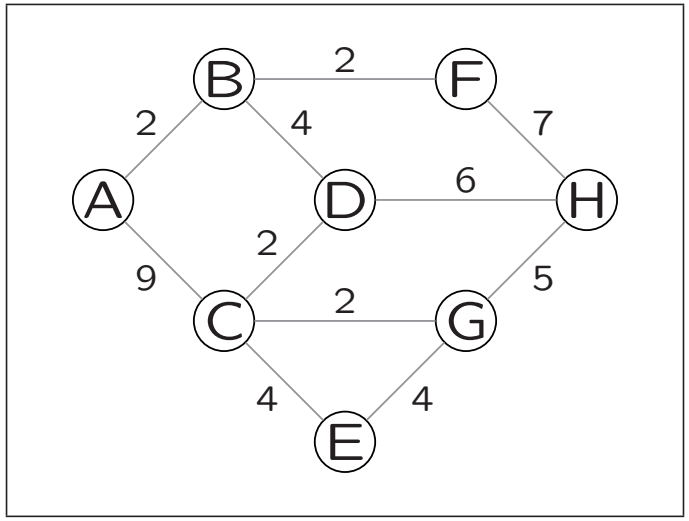
Elfde college algoritmiek

28 april 2016

Gretige Algoritmen,
Algoritme van Dijkstra,
Branch & Bound

Gegeven een samenhangende, ongerichte graaf G met gewichten op de takken.

Gevraagd: een opspannende boom van G met minimaal totaal gewicht.



// invoer: samenhangende, ongerichte gewogen graaf $G = (V, E)$
// uitvoer: E_T, verzameling takken van minimale opspannende boom T van G

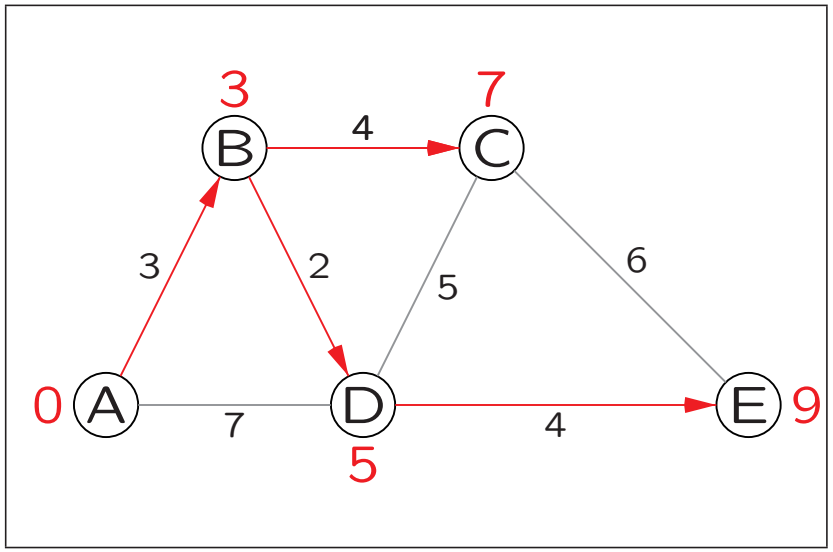
sorteer E op gewicht (in oplopende volgorde): $e_{i_1}, e_{i_2}, \ldots, e_{i_m}$;
$E_T = \emptyset$;
takteller = 0;
k = 0;
while takteller $< |V| - 1$ **do**
    k = k+1;
    if $E_T \cup \{e_{i_k}\}$ is acyclisch **then**
        $E_T = E_T \cup \{e_{i_k}\}$;
        takteller = takteller+1;
    fi
do

Complexiteit. . .
Correctheid. . .

Gegeven een graaf G met gewichten op de takken, en een beginknoop s. We nemen aan dat alle gewichten ≥ 0 zijn.

Gevraagd: voor elke willekeurige knoop v in de graaf (de lengte van) het/een kortste pad van s naar v.

Merk op dat al deze kortste paden vanuit s samen een boomstructuur vormen.



De kortste paden vanuit A zijn:

A $\longrightarrow$ B: lengte 3

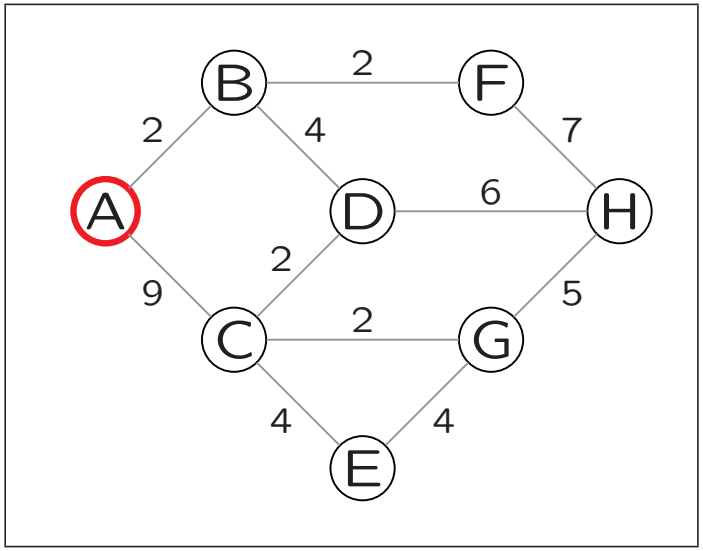
A $\longrightarrow$ B $\longrightarrow$ D: lengte 5

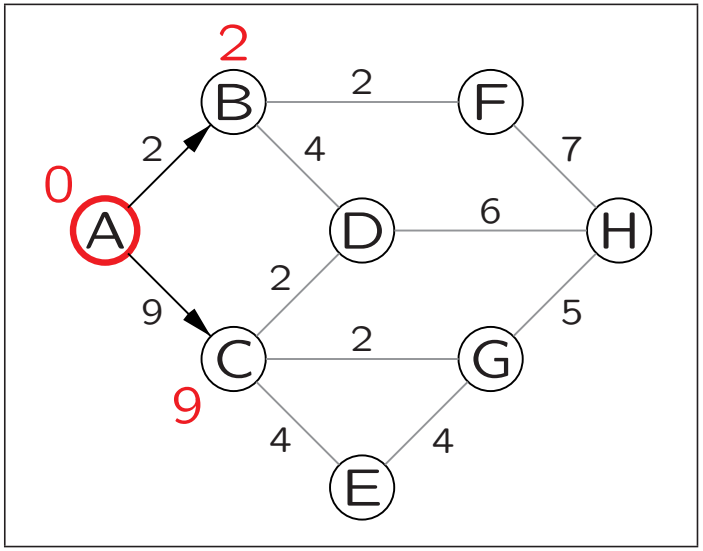
A $\longrightarrow$ B $\longrightarrow$ C: lengte 7

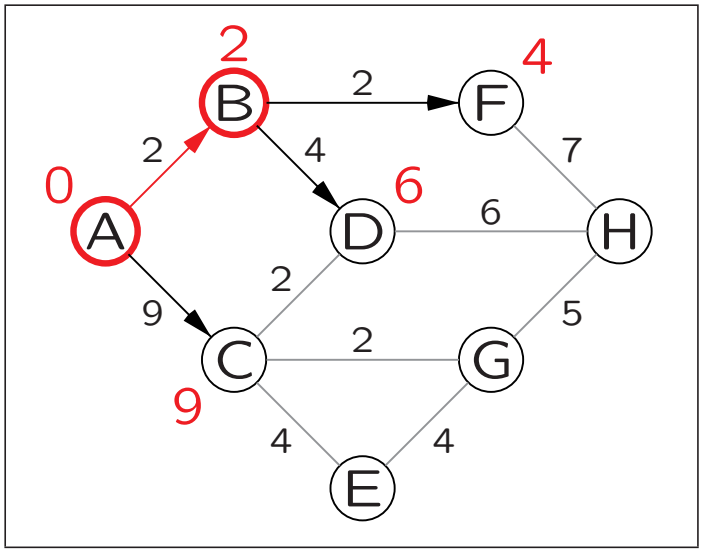
A $\longrightarrow$ B $\longrightarrow$ D $\longrightarrow$ E: lengte 9

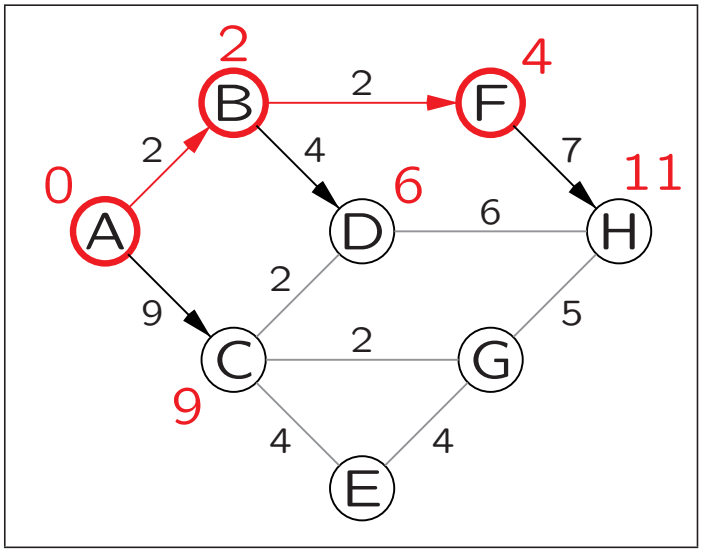
Oplossing: het algoritme van Dijkstra is een gretig algoritme, dat de kortste paden van s naar elk van de andere knopen vindt in volgorde van hun lengte. In elke stap wordt een knoop (en daarmee het pad van s naar die knoop) gekozen waarvoor het tot nu toe bekende pad vanaf s zo kort mogelijk is.

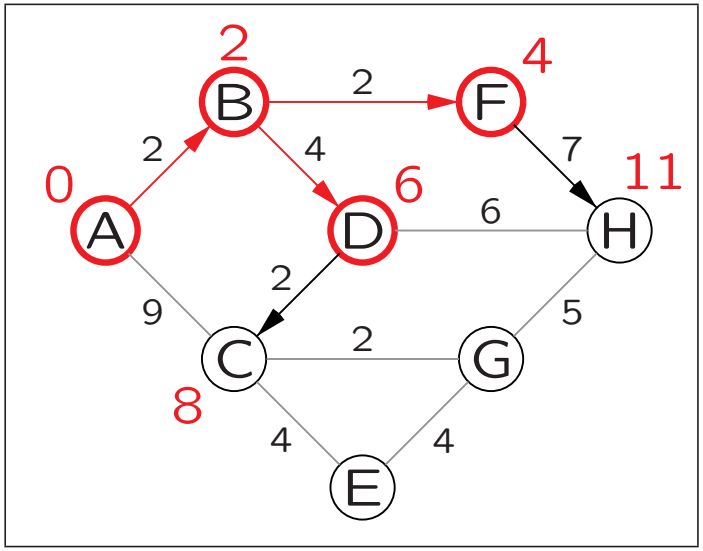
Edsger Wybe Dijkstra (Rotterdam, 11 mei 1930 — Nuenen, 6 augustus 2002) was een Nederlandse wiskundige en informaticus die veel voor de informatica heeft gedaan, met name op het gebied van gestructureerd programmeren. In 1972 werd hij onderscheiden met de Turing Award.

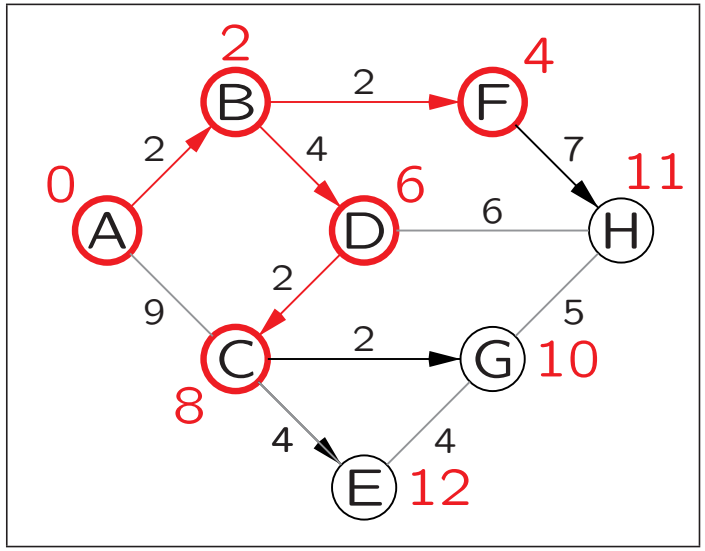

A
B
C
D
E
F
G
H
2
2
9
4
2
6
7
2
4
5

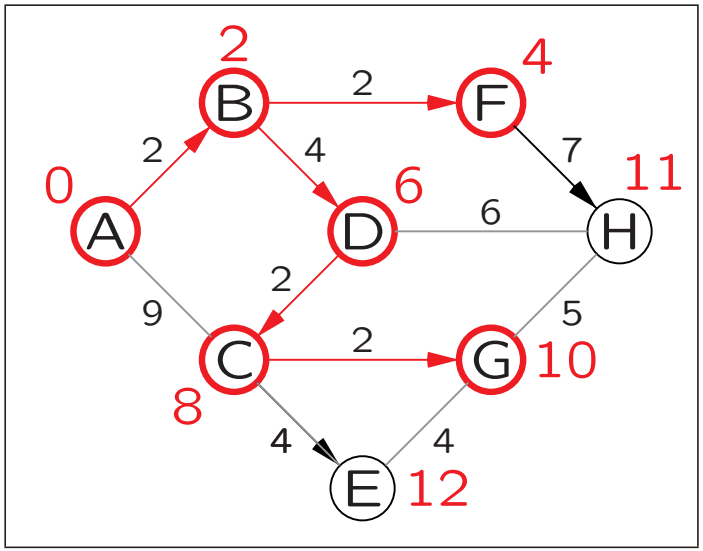

0
2
9
A
B
C
D
E
F
G
H
2
2
4
2
6
7
2
4
5

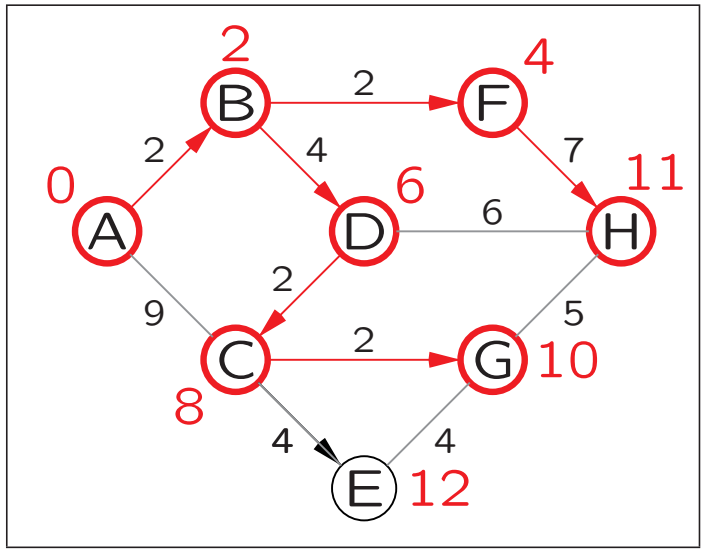

0
2
9
6
2
4
A
B
C
D
E
F
G
H
2
2
4
2
6
7
2
4
5

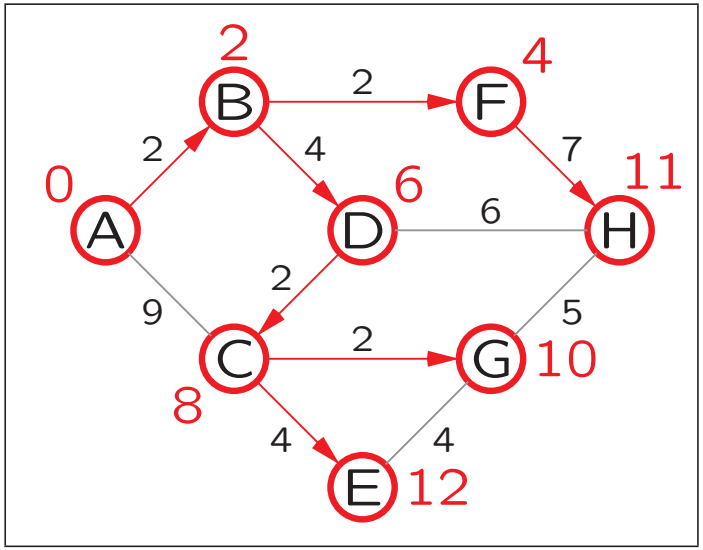

0
2
9
6
2
4
11
A
B
C
D
E
F
G
H
2
2
4
2
6
7
2
4
5


0
2
4
6
11
8
A
B
F
D
H
C
G
E
2
2
4
7
2
6
9
2
5
4
4

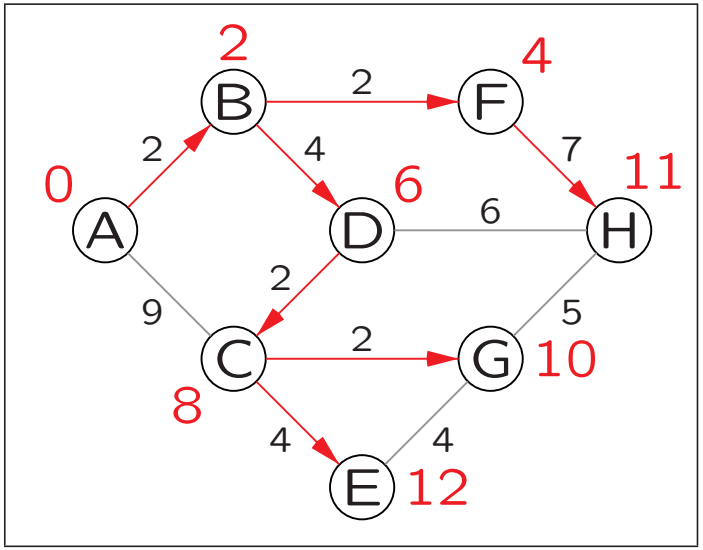

0
2
4
6
11
10
8
12
A
B
F
D
H
C
G
E
2
2
4
7
2
6
9
2
5
2
4
4


0
2
4
6
11
10
8
12
A
B
F
D
H
C
G
E
2
2
4
7
2
6
9
2
5
2
4
4


0
2
4
6
11
10
8
12
A
B
F
D
H
C
G
E
2
2
4
7
2
6
9
2
5
2
4
4



Het algoritme is klaar: alle knopen gehad



Alle kortste paden vanuit A met hun lengtes

Als je gewoon wilt doortekenen in één plaatje. . .

Begin met A, afstand 0.

A $\rightarrow$ B: $0 + 2 = 2$. OK.

A $\rightarrow$ C: $0 + 9 = 9$. OK.

Kies B, afstand 2, vanaf A.

B $\rightarrow$ D: $2 + 4 = 6$. OK.

B $\rightarrow$ F: $2 + 2 = 4$. OK.

Kies F, afstand 4, vanaf B.

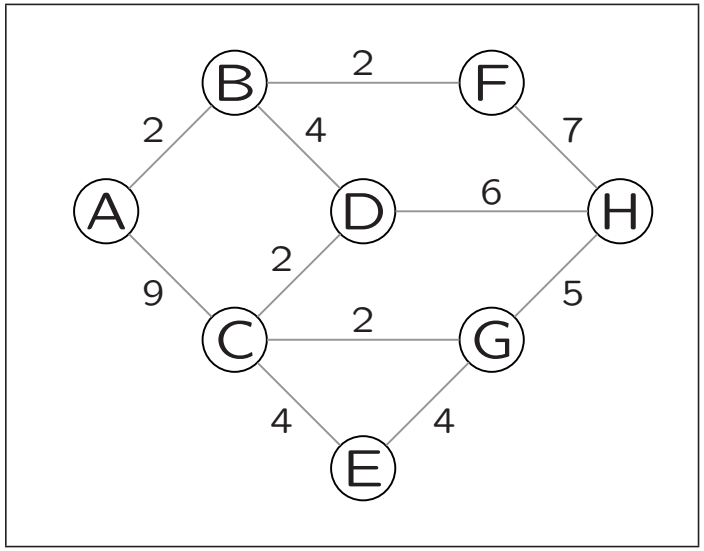
F $\rightarrow$ H: $4 + 7 = 11$. OK.

Kies D, afstand 6, vanaf B.

D $\rightarrow$ C: $6 + 2 = 8 < 9$. OK.

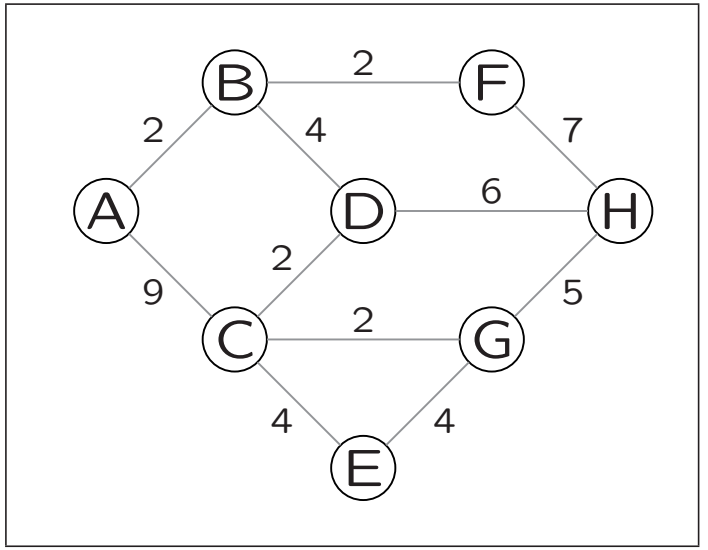
D $\rightarrow$ H: $6 + 6 = 12 > 11$. X.

Enzovoort.



Als je gewoon wilt doortekenen in één plaatje. . . (alternatief)

A	B	C	D	E	F	G	H	Actie
0	∞	∞	∞	∞	∞	∞	∞	Begin met A
–	2	9	∞	∞	∞	∞	∞	Kies B, vanaf A
–	–	9	6	∞	4	∞	∞	Kies F, vanaf B
–	–	9	6	∞	–	∞	11	Kies D, vanaf B
–	–	8	–	∞	–	∞	11	Kies C, vanaf D
–	–	–	–	12	–	10	11	Kies G, vanaf C
–	–	–	–	12	–	–	11	Kies H, vanaf F
–	–	–	–	12	–	–	–	Kies E, vanaf C



Een rij in de tabel komt overeen met het array `pad` in pseudo-code op volgende slide.

// invoer: samenhangende gewogen graaf $G = (V, E)$ en startknoop s
// uitvoer: array dat de lengtes van de kortste paden vanuit s bevat;
// na afloop is pad[v] = de lengte van een kortste pad van s naar v

for $v \in V$ **do**
    pad[v] := ∞;
od
pad[s] := 0;
$U := \emptyset$;

while ($U \neq V$) **do**
    vind knoop $v^* \in V \setminus U$ met pad[v^*] minimaal;
    $U := U \cup \{v^*\}$;
    for alle knopen v aangrenzend aan v^* **do**
        if pad[v^*] + gewicht(v^*, v) < pad[v] **then**
            pad[v] := pad[v^*] + gewicht(v^*, v);
        fi
    od
od

Complexiteit. . .

(Incorrecte!) opsplitsing while-lus:

```
for v ∈ V do
        pad[v] := ∞;
od
pad[s] := 0;
U := ∅;

while ( U ≠ V ) do
        vind knoop v* ∈ V \ U met pad[v*] minimaal;
        U := U ∪ {v*};
od
while ( U ≠ V ) do
        for alle knopen v aangrenzend aan v* do
             if pad[v*] + gewicht(v*, v) < pad[v] then
                  pad[v] := pad[v*] + gewicht(v*, v);
             fi
        od
od
```

Complexiteit met adjacency list. . .

- In het algoritme bevat U steeds alle knopen waarvan de definitieve kortste afstand vanaf s reeds bepaald is. Voor deze knopen geeft het label pad[v] al de definitieve kortste afstand aan. Moet bewezen worden.
- Voor de andere knopen w geldt na elke ronde (= doorgang door de while):

  $$(\#)\ \text{pad}[w] = \min_{u \in U}\{\text{pad}[u] + \text{gewicht}(u, w)\}^*$$

  Dit volgt direct uit het algoritme.
- De volgende dichtstbijzijnde knoop v^* wordt gekozen uit de knopen uit $V \setminus U$ die direct grenzen aan U. Nadat deze gekozen is worden de labels aangepast, zodat (#) ook geldt voor de nieuwe U.
- Het is niet zo moeilijk dit algoritme aan te passen zodat ook de kortste paden zelf worden berekend. Sla direct na het aanpassen van het label van knoop v de nieuwe kandidaattak (v^*, v) op:

  if pad[v^*] + gewicht(v^*, v) < pad[v] **then**
  pad[v] := pad[v^*] + gewicht(v^*, v);
  nieuwe kandidaattak voor v: (v^*, v)
  fi

*Dit betekent dat pad[w] voor deze knopen $w \notin U$ de lengte van een kortste pad van s naar w aangeeft via uitsluitend knopen van U.

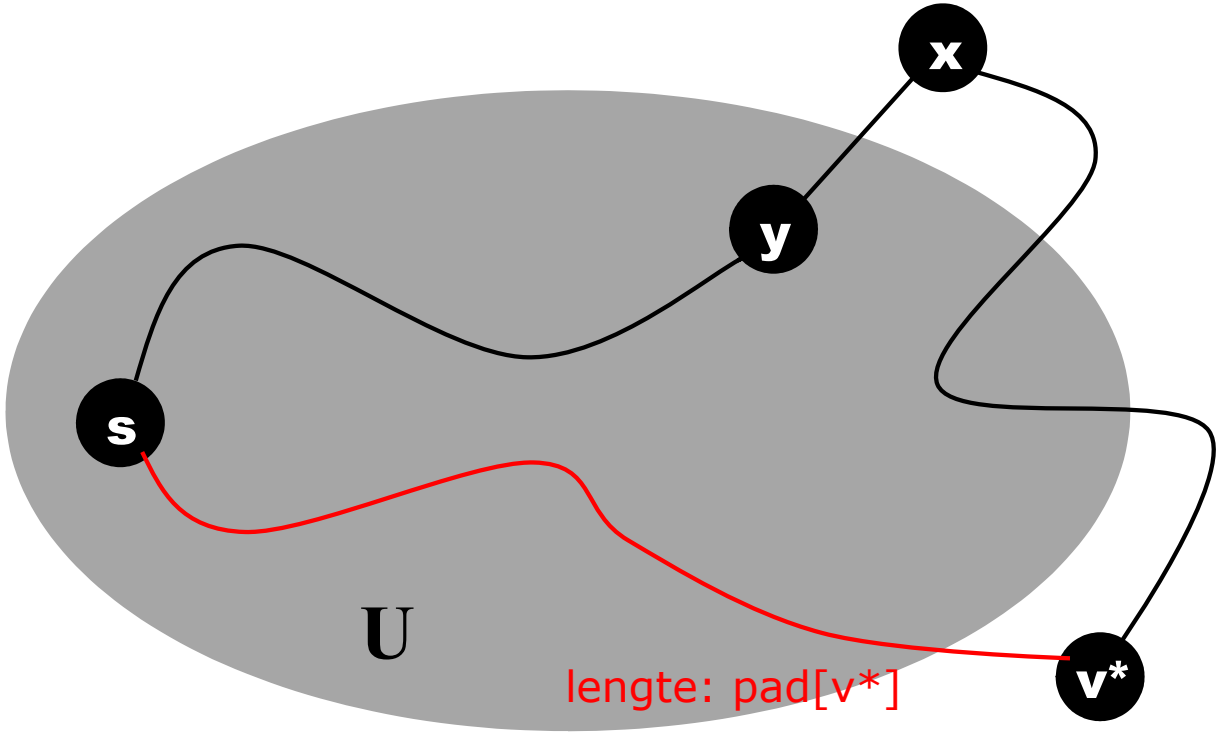
Na elke ronde (dus ook na de laatste, wanneer $U = V$) van het algoritme van Dijkstra geldt:

- U bevat alle knopen waarvan de definitieve kortste afstand vanaf s reeds bepaald is. Voor elke $v \in U$ geeft het label pad[v] die kortste afstand aan.

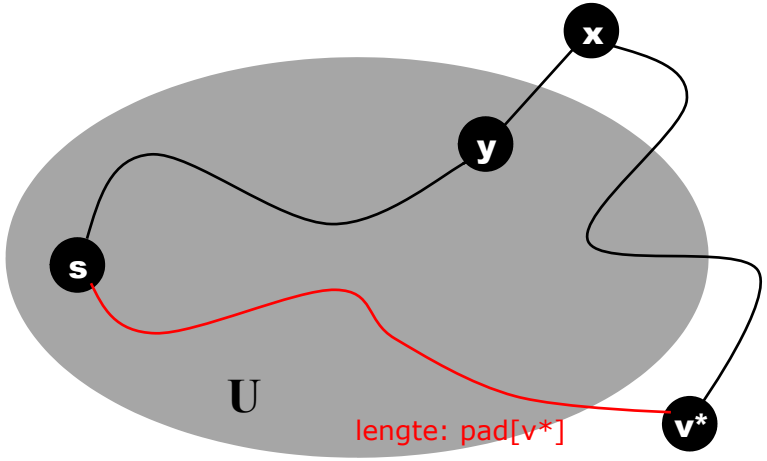
Om dit te bewijzen moet je laten zien dat:

- wanneer v^* wordt toegevoegd aan U, pad[v^*] inderdaad de lengte van het kortste pad van s naar v^* bevat

Essentie van het bewijs



Bekijk, behalve het pad ter lengte pad[v^*] van s naar v^* via U een willekeurig ander pad van s naar v^*. Stel dat x de eerste knoop op dat pad is buiten U, en y de laatste knoop daarvóór. (Deze x kán gelijk zijn aan v^*.) Zij d(s, y, x, v^*) de lengte van dat pad.



Dan geldt: d$(s, y, x, v^*) \geq$ d$(s, y, x) \geq$ pad$[x] \geq$ pad$[v^*]$ omdat respectievelijk alle gewichten van de takken ≥ 0 zijn, (#) geldt voor x en v^* gekozen was in deze ronde als "minimale" knoop.

QED

Backtracking

- bouwt een oplossing component voor component op
- kijkt tijdens de stap-voor-stap constructie of de deeloplossing die bekeken wordt nog aan de gestelde restricties/eisen voldoet (kan voldoen)
- zo niet, breidt dan de deeloplossing niet verder uit en herziet de laatste keuze (backtrack!)
- houdt (alleen) het pad corresponderend met de (deel)oplossing die 'nu' wordt uitgebreid bij
- spaart soms veel werk uit vergeleken met exhaustive search
- toepasbaar op allerlei problemen waarbij oplossingen stap voor stap gegenereerd kunnen worden
- werking te beschrijven m.b.v. een state space tree

Optimalisatieprobleem: er wordt een oplossing gezocht met minimale of maximale .....

Terminologie:

- De functie/waarde die ge-optimaliseerd moet worden heet wel de objectfunctie (bijvoorbeeld de lengte van een Hamiltonkring)
- Een oplossing die voldoet aan de restricties van het probleem heet feasible (toelaatbaar)
- Een optimale oplossing is een/de toelaatbare oplossing met de beste waarde van de objectfunctie

Backtracking en optimalisatieproblemen

- Bij een minimalisatieprobleem kun je bijv. ook stoppen met uitbreiden wanneer je deeloplossing al een grotere waarde van de te optimaliseren functie heeft dan de huidige beste oplossing (die wordt dus bijgehouden/opgeslagen) en alleen maar nog groter kan worden. (Iets dergelijks bij een maximalisatieprobleem.)

- Als je snel een (bijna) optimale oplossing vindt, kunnen verdere deeloplossingen eerder worden afgekapt en ben je dus sneller klaar.

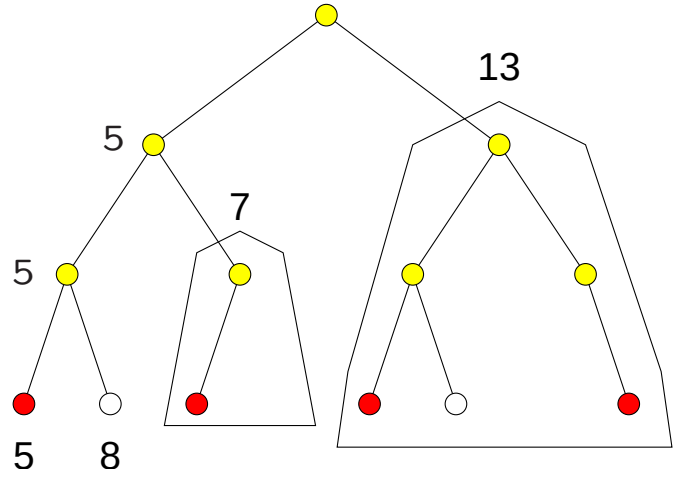
- Backtracking is het efficiëntst wanneer *een* oplossing gezocht wordt, dus voor optimalisatieproblemen is backtracking niet bij voorbaat de meest geschikte oplossingsmethode.

Assignmentprobleem (toewijzingsprobleem)

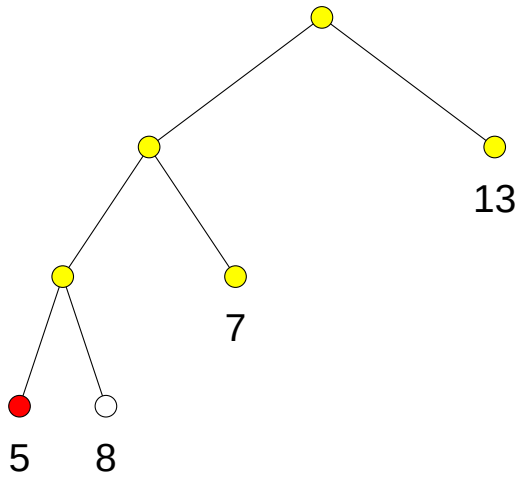
Gegeven n personen en n taken (jobs). Persoon i kan taak j doen voor `kosten`$[i][j]$ euro. **Gevraagd**: de/een toewijzing van de personen aan de jobs (één persoon per job en één job per persoon) met minimale kosten.

Voorbeeld:

	W	X	Y	Z
Alice	9	2	7	8
Bob	6	4	3	7
Carol	5	8	1	8
David	7	6	9	4



state space tree



gesnoeide boom

Witte knopen corresponderen met toelaatbare oplossingen; rode knopen met niet-toelaatbare oplossingen; de waarden bij de bladeren geven de waarde van de objectfunctie van de bijbehorende oplossing; de waarden bij de andere knopen geven een ondergrens op de te verwachten waarde van de objectfunctie; bij de knoop met grens 13 kan met-een gesnoeid worden, die met 7 wordt nog bekeken, maar leidt tot een niet-toelaatbare oplossing

	W	X	Y	Z
Alice	9	2	7	8
Bob	6	4	3	7
Carol	5	8	1	8
David	7	6	9	4

Een ondergrens voor de kosten van een optimale oplossing:

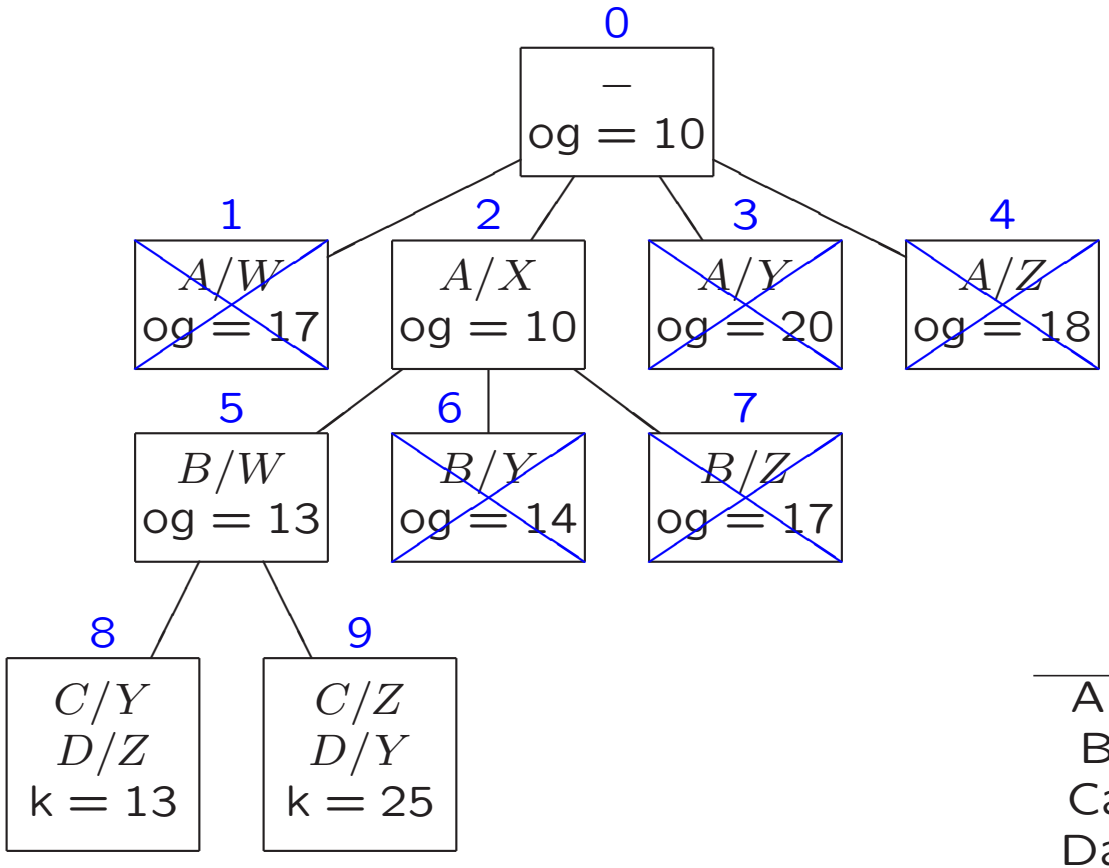
(a) neem uit elke rij de kleinste waarde en tel die bij elkaar op: $2+3+1+4=10$

(b) neem uit elke kolom de kleinste waarde en tel die bij elkaar op: $5+2+1+4=12$

We genereren oplossingen door de personen een voor een aan een job te koppelen, en gebruiken daarbij de ondergrens volgens suggestie (a).
De optimale oplossing heeft totale kosten 13:
Alice job X; Bob job W; Carol job Y; David job Z

Voor een willekeurige knoop/deeloplossing berekenen we de ondergrens op de te verwachten waarde van de objectfunctie door uit elke verdere rij de kleinste waarde van de nog beschikbare jobs te nemen en deze op te tellen bij de waarde van de deeloplossing. Voor de deeloplossing(en) die Alice aan job W koppelt zal die ondergrens bijvoorbeeld $9+3+1+4 = 17$ zijn. (Analoog voor kolommen: kies uit elke verdere kolom de kleinste waarde van de nog beschikbare personen(*).)

De volgorde waarin de knopen van de state space tree worden uitgebreid laten we afhangen van de berekende ondergrens. We kiezen de knopen met de beste (=laagste) ondergrens als eerste: dit lijkt de meest veelbelovende knoop. Deze strategie heet wel de best-fit-first branch-and-bound.



	W	X	Y	Z
Alice	9	2	7	8
Bob	6	4	3	7
Carol	5	8	1	8
David	7	6	9	4

Opgave: los het probleem op met de ondergrenzen berekend volgens (*).

- **Lezen/leren bij dit college:**
  paragraaf 9.2 t/m blz 353, paragraaf 9.3, slides, paragraaf 12.2

- **Werkcollege:**
  donderdag 28 april 2016, 13:45–15:30, in zaal Paleistuin:
  derde programmeeropdracht (dynamisch programmeren)

- **Volgend college:**
  donderdag 12 mei 2016