Elfde college algoritmiek

3 mei 2012

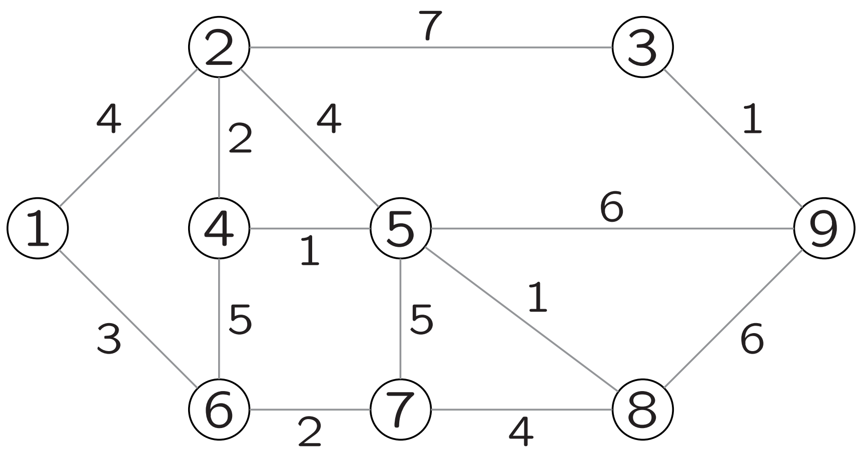
Gretige algoritmen en Branch & Bound

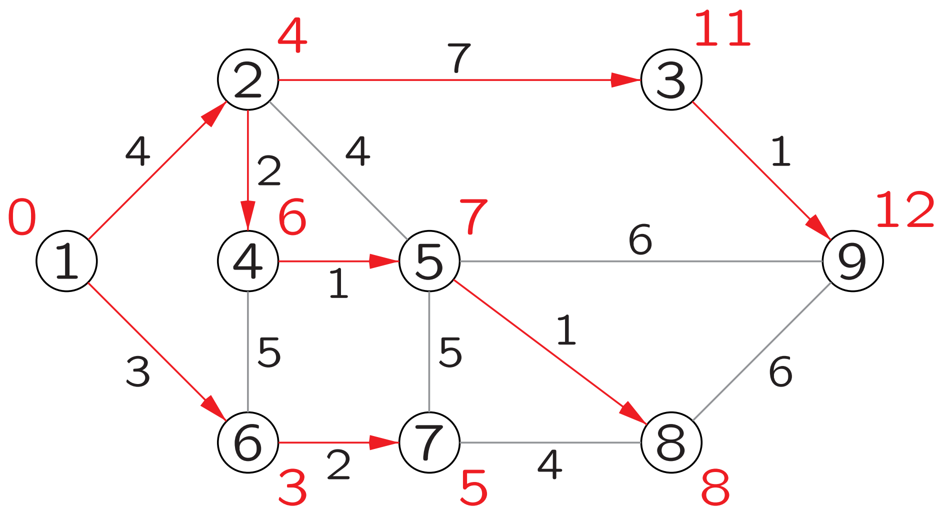
Gegeven een graaf G met gewichten op de takken, en een beginknoop s. We nemen aan dat alle gewichten ≥ 0 zijn.

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.

Voorbeeld

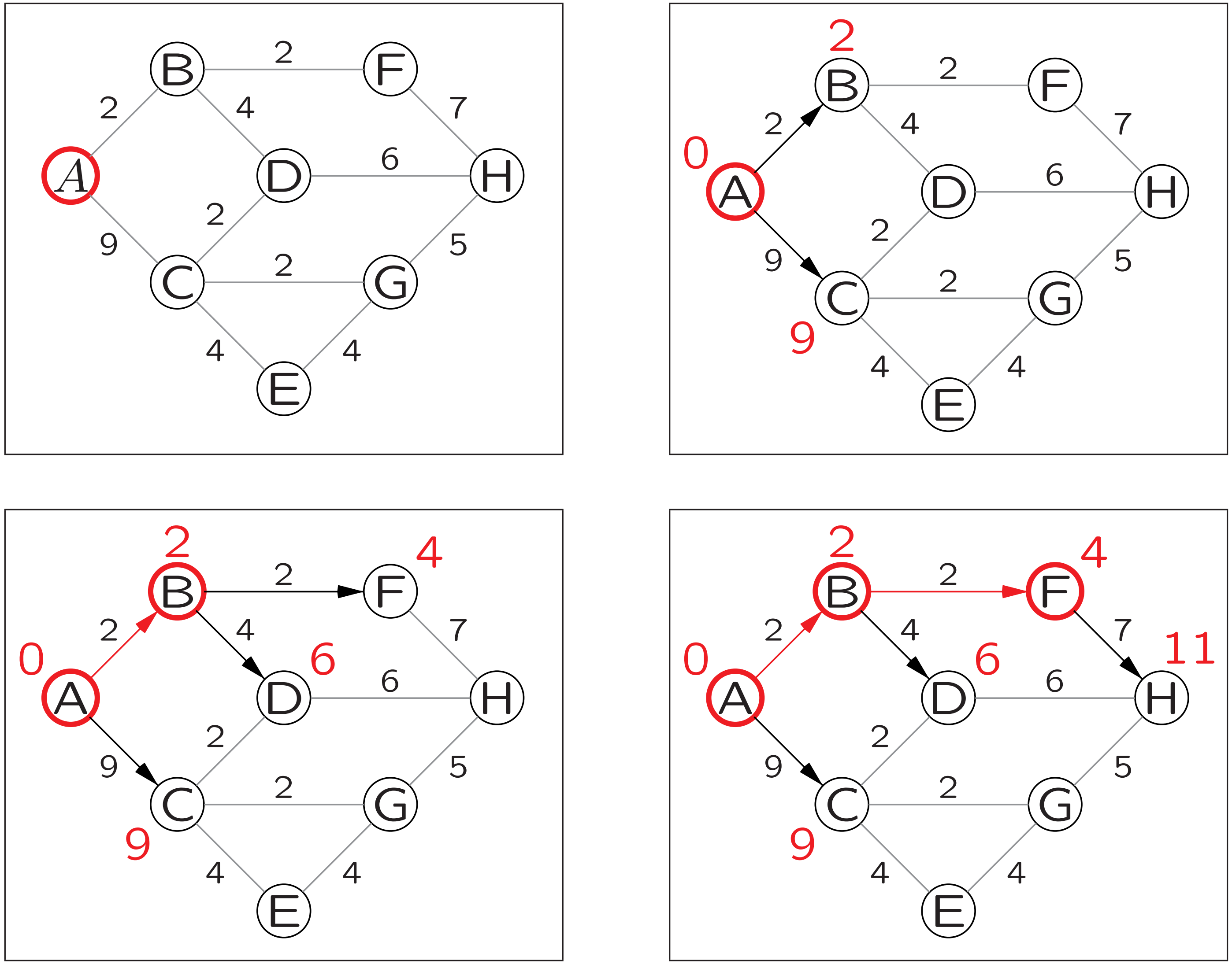

1
2
3
4
5
6
7
8
9
7
4
2
4
1
1
6
3
5
5
1
6
2
4


0
4
11
6
7
12
3
5
8
1
2
3
4
5
6
7
8
9
7
4
2
4
1
1
6
3
5
5
1
6
2
4

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.

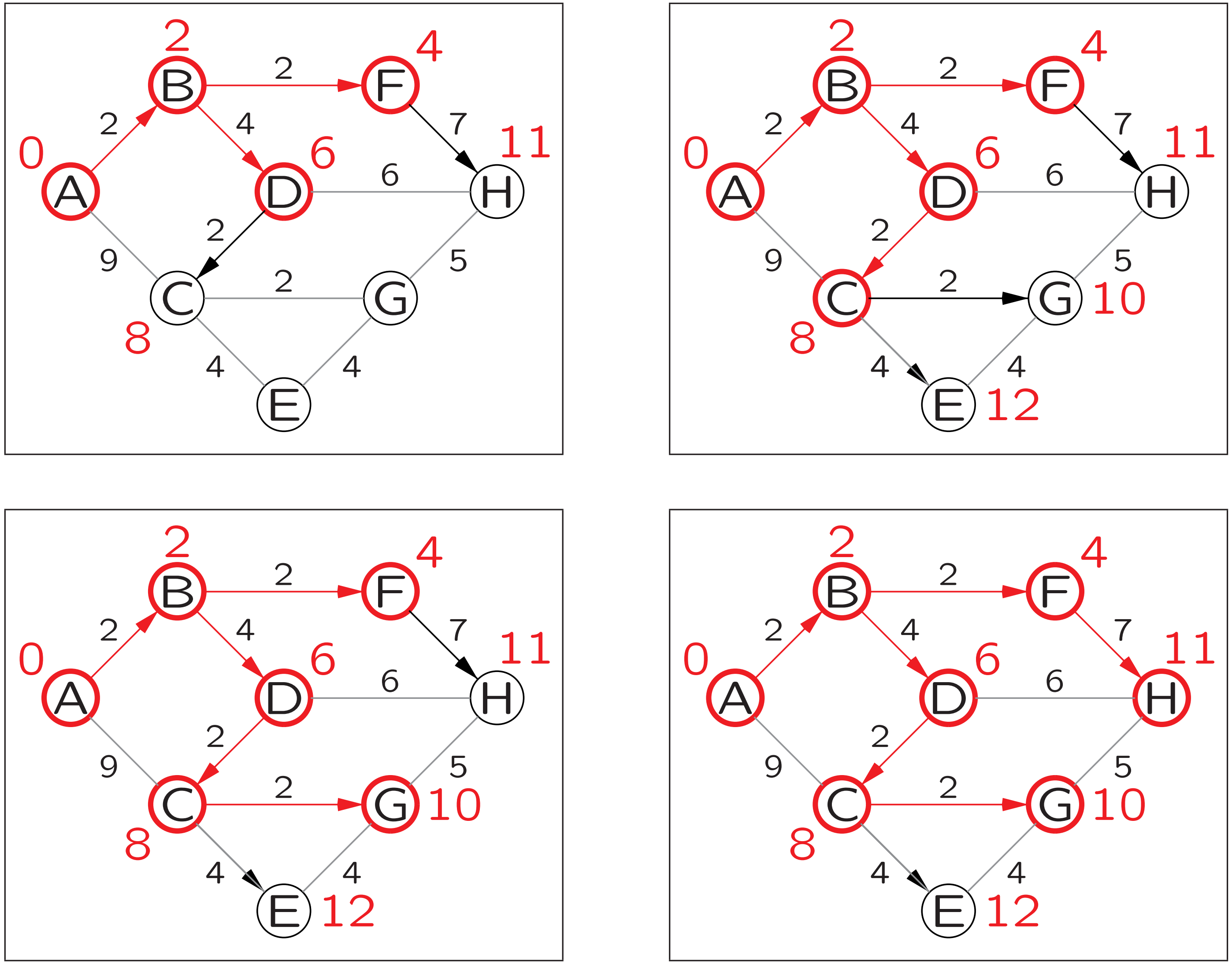
Dijkstra: Edsger W. Dijkstra (1930-2002)

Voorbeeld -1-

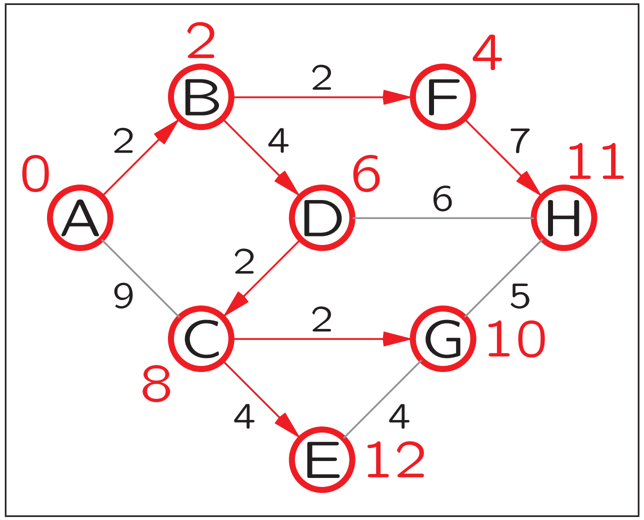


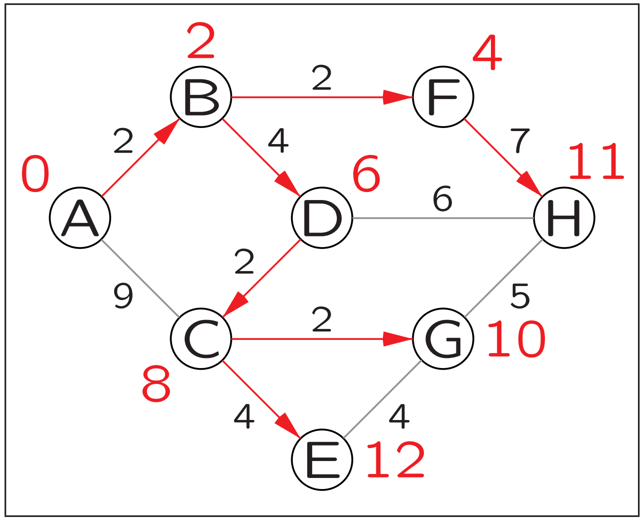
B 2 F
2 4 7
A D 6 H
2
9 C 2 G 5
4 4
E
2
B 2 F
2 4 7
0 A D 6 H
2
9 C 2 G 5
9
4 4
E
2 4
B 2 F
2 4 7
0 A 6 D 6 H
2
9 C 2 G 5
9
4 4
E
2 4
B 2 F
2 4 7 11
0 A 6 D 6 H
2
9 C 2 G 5
9
4 4
E

Voorbeeld -2-



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





Het algoritme is klaar: alle knopen gehad

Alle kortste paden vanuit A met hun lengtes

// 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] $:= \infty$; **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...

- 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 afstand aan. Moet bewezen worden.
- Voor de andere knopen w geeft pad[w] steeds de kortste afstand vanuit s naar w via uitsluitend knopen uit U. Moet bewezen worden.
- De volgende dichtstbijzijnde knoop wordt gekozen uit de knopen uit $V \setminus U$ die direct grenzen aan U. Nadat deze gekozen is worden de labels aangepast.
- In elke stap wordt zo voor elke knoop uit $V \setminus U$ de kortste afstand vanaf s via de reeds afgehandelde knopen uit U (en bijbehorende takken) bepaald.
- Het is niet zo moeilijk dit algoritme aan te passen zodat ook de kortste paden zelf worden berekend.

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

1. 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.
2. Voor de andere knopen w ($w \notin U$) is pad[w] de kortste afstand vanuit s naar w via uitsluitend knopen uit U.

Om 1. en 2. te bewijzen moet je laten zien dat:

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

b. na elke update van de labels na toevoeging van v^* de pad[w] nog steeds de kortste afstand via (de nieuwe) U aangeven, voor alle $w \notin U$

Uit werkcollege 10:

Opgave 7. (zie tweede editie, opgave 9.3.3)

Geef een voorbeeld waaruit blijkt dat het algoritme van Dijkstra niet altijd werkt voor een samenhangende gewogen graaf met negatieve gewichten. Dat wil zeggen: geef een samenhangende gewogen graaf met minstens één negatief gewicht, waarbij het algoritme van Dijkstra vanuit een bepaalde startknoop niet voor alle andere knopen het kortste pad vindt.

Een graaf met 3 knopen is al genoeg.

Gegeven een verzameling $A = \{1, 2, \ldots, n\}$ met activiteiten die allemaal gebruik willen maken van een of andere "resource" (bijvoorbeeld een collegezaal). Deze resource kan maar door één activiteit tegelijk gebruikt worden. Activiteit i vindt plaats gedurende het tijdsinterval $[b_i, e_i)$. De begin- en eindtijden b_i en e_i zijn voor alle activiteiten bekend.

Definitie: activiteiten i en j heten compatibel als $[b_i, e_i)$ en $[b_j, e_j)$ niet overlappen, dus als $e_i \leq b_j$ of $e_j \leq b_i$.

Opdracht: vind een zo groot mogelijke deelverzameling van paarsgewijs compatibele activiteiten uit A.

Gretig algoritme: in elke stap

- wordt een activiteit i gekozen die compatibel is met de reeds gekozen activiteiten en die op dat moment het beste lijkt (gretige keuze)

Mogelijke gretige keuzes voor het kiezen van activiteit i:

1. selecteer steeds de activiteit die het eerst begint
2. selecteer steeds de activiteit die het kortste duurt
3. selecteer steeds de activiteit die overlapt met zo min mogelijk andere activiteiten
4. selecteer steeds de activiteit die het eerst eindigt

Welke gretige keuzes voor het kiezen van activiteit i leiden altijd tot een optimale oplossing?

1. selecteer de activiteit die het eerst begint: werkt niet

2. selecteer de activiteit die het kortste duurt: werkt niet

3. selecteer de activiteit die overlapt met zo min mogelijk andere activiteiten: werkt niet

4. selecteer de activiteit die het eerst eindigt: werkt wel

```
// neem aan dat A oplopend gesorteerd is op eindtijd e_i,
// anders eerst even sorteren: O(n lg n)
A' := {1};
j := 1;
// de laatst aan A' toegevoegde activiteit
for i := 2 to n do
// loop activiteiten af in volgorde van eindtijd
    if i is compatibel met A' then (*)
       A' := A' ⋃ {i}; j := i;
    fi
od
// A' bevat nu een optimale paarsgewijs
// compatibele deelverzameling van A
```

Correctheid: moet bewezen worden
Complexiteit: $\mathrm{O}(n)$ als A reeds gesorteerd is

i	b_i	e_i
1	1	4
2	3	5
3	0	6
4	5	7
5	3	8
6	5	9
7	6	10
8	8	11
9	8	12
10	2	13
11	12	14

Optimale oplossing: $\{1, 4, 8, 11\}$

De **correctheid** van het gretige algoritme volgt uit de volgende twee observaties:

1. Er bestaat een optimale oplossing die met activiteit 1 begint (die met de kleinste eindtijd dus).

2. Stel A' is een optimale oplossing van het oorspronkelijke probleem, dus met activiteitenverzameling A, die 1 bevat. Dan is $B' = A' \setminus \{1\}$ een optimale oplossing van het probleem met activiteitenverzameling $B = \{i \in A : b_i \geq e_1\}$.

```
// neem aan dat A oplopend gesorteerd is op eindtijd e_i,
// anders eerst even sorteren: O(n lg n)
A' := {1};
j := 1;
// de laatst aan A' toegevoegde activiteit
for i := 2 to n do
// loop activiteiten af in volgorde van eindtijd
    if i is compatibel met A' then (*)
       A' := A' ⋃ {i}; j := i;
    fi
od
// A' bevat nu een optimale paarsgewijs
// compatibele deelverzameling van A
```

(*) Merk op: i is compatibel met A' als hij compatibel is met de laatst toegevoegde activiteit.

Dus (∗) wordt: `if` $b_i \geq e_j$ `then`

Correctheid: moet bewezen worden
Complexiteit: $O(n)$ als A reeds gesorteerd is

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 <u>de</u> (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:

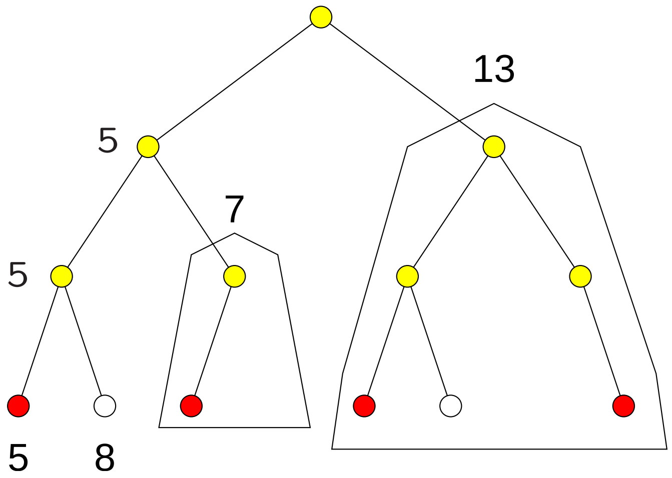
- De functie/waarde die ge-optimaliseerd moet worden heet wel de objectfunctie (bijvoorbeeld de lengte van een Hamiltonkring)

- Een oplossing die voldoet aan de restricties van het probleem heet feasible (toelaatbaar)

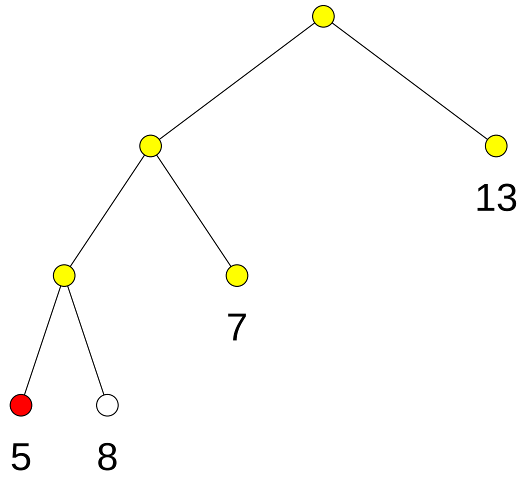
- Een optimale oplossing is een/de toelaatbare oplossing met de beste waarde van de objectfunctie

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.)
- Backtracking is het efficiëntst wanneer *een* oplossing gezocht wordt, dus voor optimalisatieproblemen is backtracking niet bij voorbaat de meest geschikte oplossingsmethode.
- Als je snel een (bijna) optimale oplossing vindt, kunnen verdere deeloplossingen eerder worden afgekapt en ben je dus sneller klaar.



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 meteen gesnoeid worden, die met 7 wordt nog bekeken, maar leidt tot een niet-toelaatbare oplossing

Assignmentproblem (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

	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:

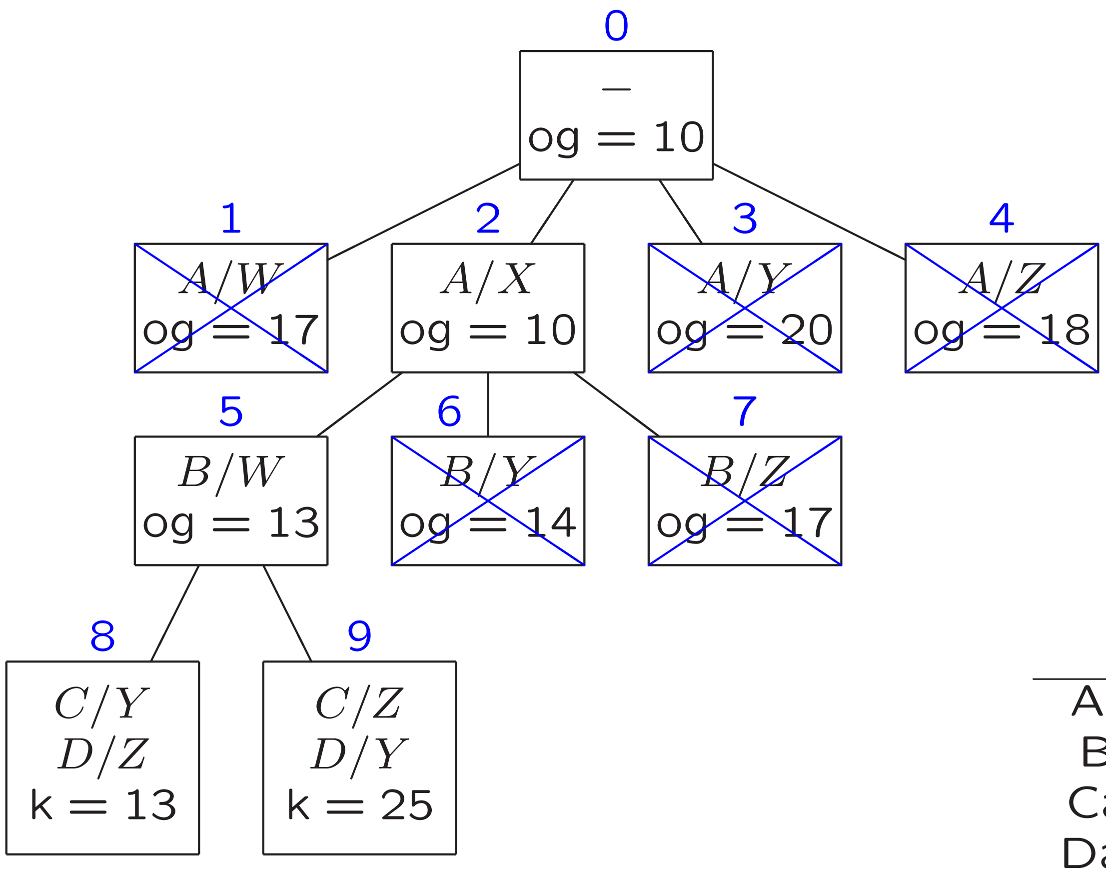
(a) neem uit elke rij de kleinste waarde en tel die bij elkaar op: $2 + 3 + 1 + 4 = 10$

(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 (*).

Branch & bound

- is alleen toepasbaar op optimalisatieproblemen
- genereert oplossingen stap voor stap en houdt de tot dusver gevonden beste oplossing bij
- gebruikt voor elke deeloplossing (= knoop in de state space tree) een of andere ondergrens (minimalisatieprobleem) resp. bovengrens (maximalisatieprobleem) op de te verwachten waarde van de objectfunctie, met als doel
  - van deeloplossingen te kunnen bepalen dat ze niet verder bekeken hoeven te worden: snoeien (*)
  - de zoekvolgorde in de zoekruimte (state space tree), dus de volgorde waarin knopen worden gegenereerd en verder bekeken, te leiden (**)

(*) zou bij backtracking ook kunnen, maar (**) niet of nauwelijks

Een branch and bound algoritme breidt een knoop (deeloplossing) niet verder uit als

- de waarde van de ondergrens (bovengrens) bij die knoop niet beter is dan de waarde van de tot dusver gevonden beste oplossing: als ondergrens $\geq$ tot dusver gevonden minimale waarde, dan snoeien

- de deeloplossing niet meer voldoet aan de restricties (of niet meer uit te breiden is tot een toelaatbare oplossing)

- er nog maar één volledige, toelaatbare oplossing mogelijk is bij de deeloplossing (i.h.b. als deeloplossing zo'n volledige oplossing *is*); de waarde van deze ene oplossing wordt met beste oplossing tot nu toe vergeleken; update zonodig beste oplossing.

De volgorde waarin de knopen (deeloplossingen) worden uitgebreid hangt direct af van de berekende grenzen:

- er worden meerdere deeloplossingen tegelijk bijgehouden, dit in tegenstelling tot backtracking
- in elke stap wordt een van al deze deeloplossingen gekozen, en daarvan worden alle 1-staps-uitbreidingen (kinderen in de state space tree) bekeken en geëvalueerd (d.w.z. ondergrens/bovengrens bepaald)
- zinloze uitbreidingen worden meteen verworpen
- meestal wordt de deeloplossing gekozen die het meest veelbelovend lijkt: best-fit-first branch-and-bound
- bij minimalisatieproblemen (resp. maximalisatieproblemen) kiezen we de knoop met de laagste ondergrens (resp. hoogste bovengrens) als eerste

Los het toewijzingsprobleem op voor onderstaand voorbeeld met behulp van

1. backtracking (snoeien op de kosten van deeloplossingen (*))
2. branch and bound

en vergelijk de hoeveelheid snoeiwerk bij beide methoden, alsmede de volgorde waarin de knopen van de state space tree worden bekeken.

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

(*) overigens kun je bij backtracking ook ondergrenzen berekenen zoals bij B&B, en die gebruiken om te snoeien; B&B vindt een optimale oplossing echter i.h.a. eerder

Gegeven n objecten, met gewicht $w_1, \ldots, w_n$ en waarde $v_1, \ldots, v_n$, en een knapzak met capaciteit W. **Gevraagd**: de meest waardevolle deelverzameling der objecten die in de knapzak past (dus met totaalgewicht $\leq W$).

Voorbeeld:

item	w	v	v/w
1	4	$40	10
2	7	$42	6
3	5	$25	5
4	3	$12	4

Maximaal gewicht $W = 10$

item	w	v	v/w
1	4	$40	10
2	7	$42	6
3	5	$25	5
4	3	$12	4

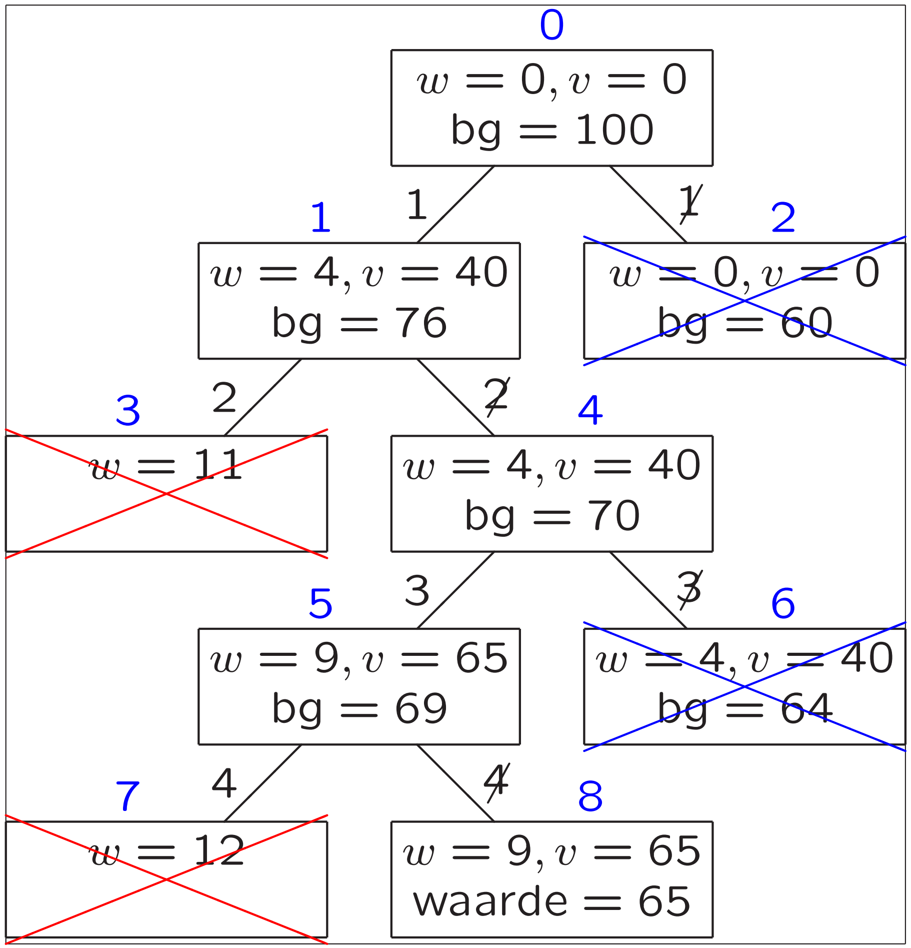
Opbouwen van de oplossing: in de i-de stap wordt object i wel of niet gekozen,

Een bovengrens voor de kosten van een optimale oplossing:

- Bij aanvang: $W * (v_1/w_1) = 100$;
- Na de i-de stap: $v + (W - w) * (v_{i+1}/w_{i+1})$, met v de totaalwaarde van de reeds gekozen objecten en w het totaalgewicht daarvan.

De optimale oplossing heeft gewicht 9 en waarde 65: $\{1, 3\}$.

B&B state space tree



item	w	v	v/w
1	4	\$40	10
2	7	\$42	6
3	5	\$25	5
4	3	\$12	4

$W = 10$

Zie ook exercise 12.2.5 **(ook in tweede editie)**, Levitin.

- **Lezen/leren bij dit college:** Paragrafen 9.3 en 12.2, sheets
- **Werkcollege:**
  donderdag 3 mei 2012, 13:45–15:30, in zaal Paleistuin:
  derde programmeeropdracht (dynamisch programmeren)
- **Opgaven:**
  zie http://www.liacs.nl/home/rvvliet/algoritmiek/
- **Volgend college:** donderdag 10 mei 2012