Elfde college algoritmiek

1 mei 2009

Dijkstra 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.

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 dus het pad van s naar die knoop) gekozen die het dichtst bij s ligt.

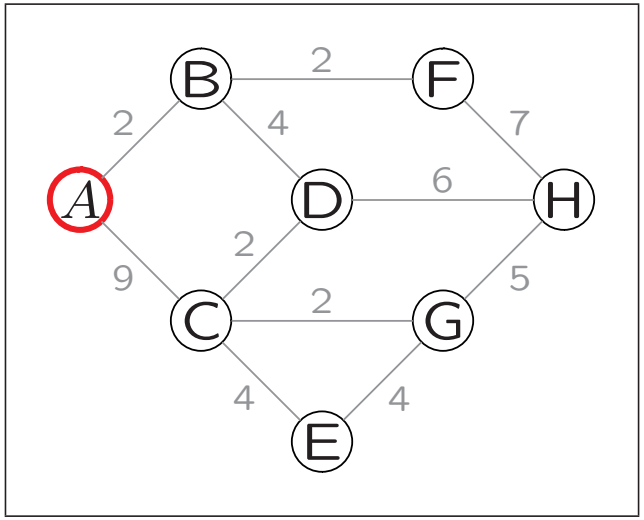
Dijkstra: Edsger W. Dijkstra (1930-2002)

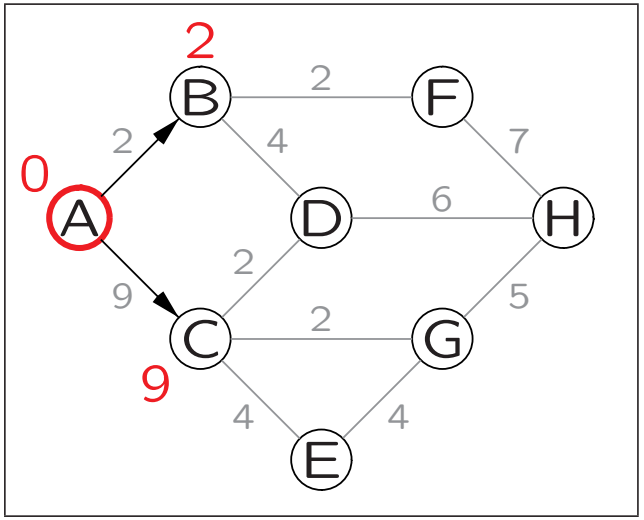
Algoritme van Dijkstra

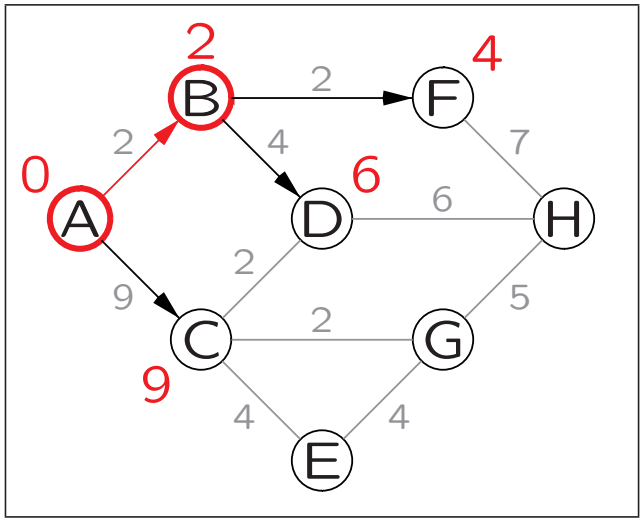
// invoer: samenhangende gewogen graaf $G = (V, E)$ en startknoop s
// uitvoer: array dat de lengtes van de kortste paden vanuit s bevat;
// na afloop is pad[v] = de lengte van een kortste pad van s naar v
for $v \in V$ **do**
    pad[v] := ∞; **od**
pad[s] := 0;
$U := \emptyset$;
while ($U \neq V$) **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

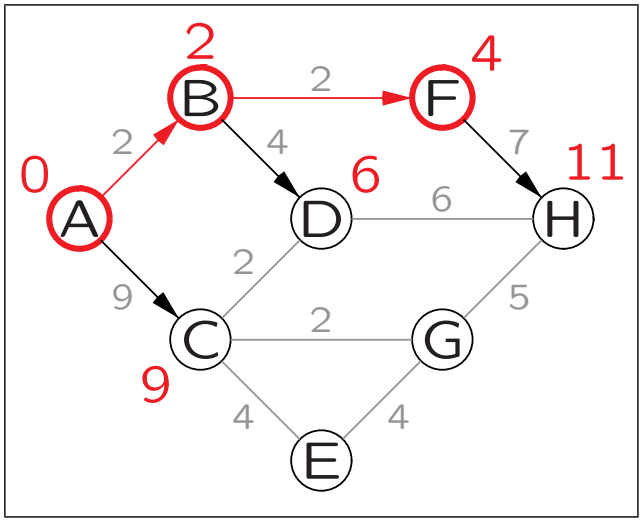
- In het algoritme bevat U steeds alle knopen waarvan de definitieve kortste afstand vanaf s reeds bepaald is. Voor deze knopen geeft het label pad[v] al de definitieve afstand aan. Moet bewezen worden.
- Voor de andere knopen w geeft pad[w] steeds de kortste afstand vanuit s naar w via uitsluitend knopen 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.

Voorbeeld -1-

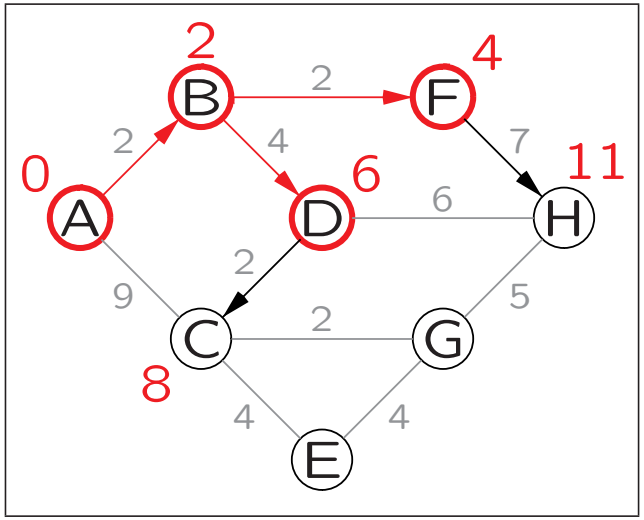

A
B
C
D
E
F
G
H
2
2
4
7
6
2
9
2
5
4
4

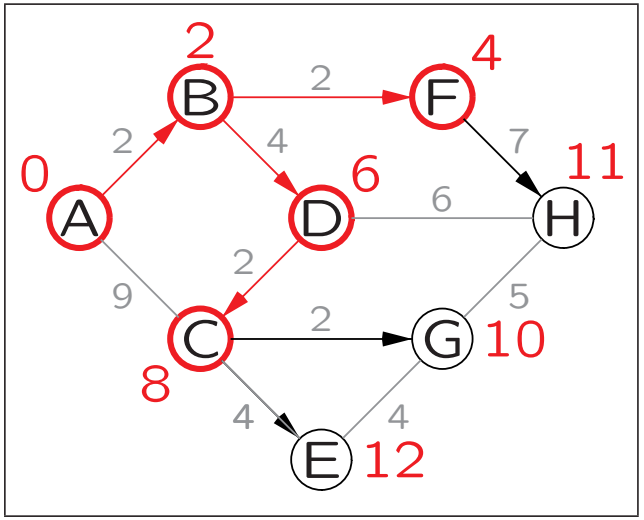

0
2
9
A
B
C
D
E
F
G
H
2
2
4
7
6
2
2
5
4
4

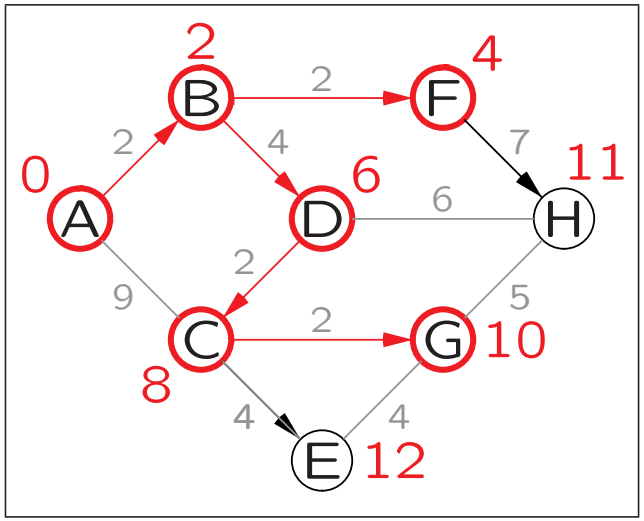

0
2
9
6
4
9
A
B
C
D
E
F
G
H
2
2
4
7
6
2
2
5
4
4

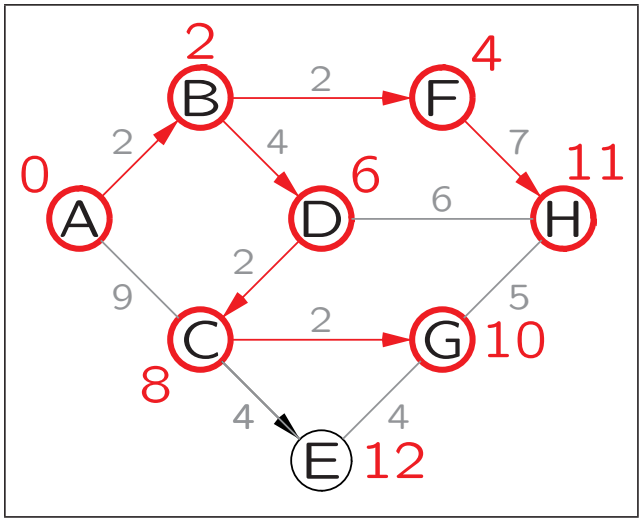

0
2
9
6
4
9
11
A
B
C
D
E
F
G
H
2
2
4
7
6
2
2
5
4
4

Voorbeeld -2-

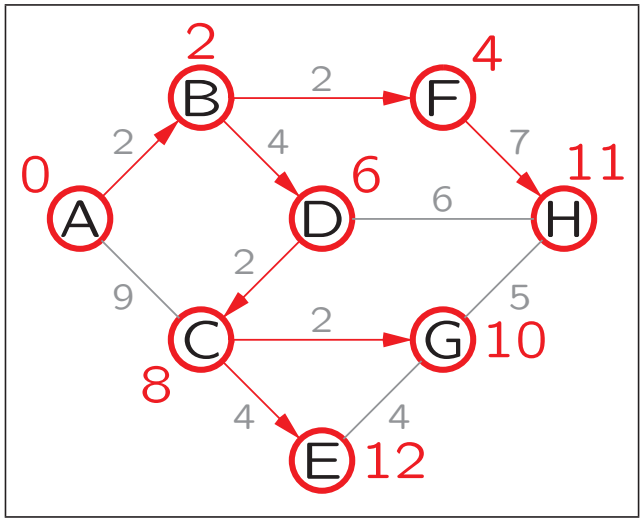

A
B
C
D
E
F
G
H
0
2
8
6
4
11
2
2
9
4
2
6
2
7
2
5
4
4


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

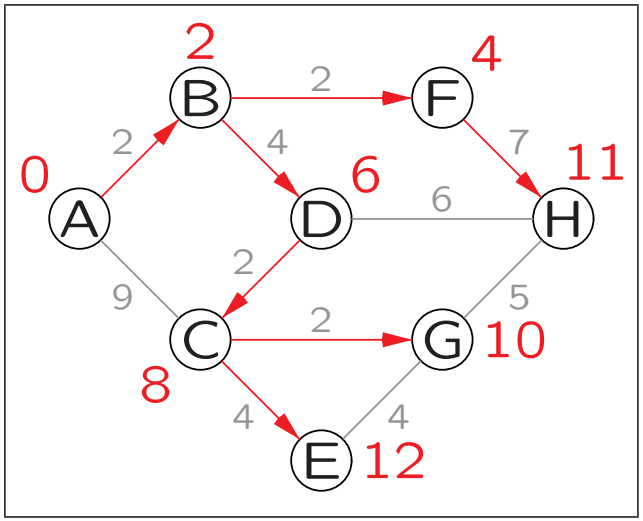

A
B
C
D
E
F
G
H
0
2
8
6
4
12
11
10
2
2
9
4
2
6
2
7
2
5
4
4


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

Voorbeeld -3-



Het algoritme is klaar:
alle knopen gehad



Alle kortste paden vanuit
A met hun lengtes

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 $w \notin U$

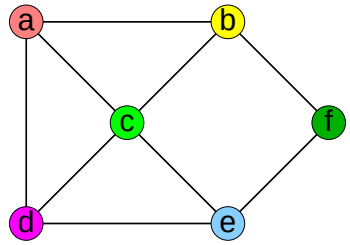
Backtracking

- 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!)
- spaart zo soms veel werk uit vergeleken met exhaustive search
- toepasbaar op allerlei problemen waarbij oplossingen stap voor stap gegenereerd kunnen worden

Voorbeeldprobleem

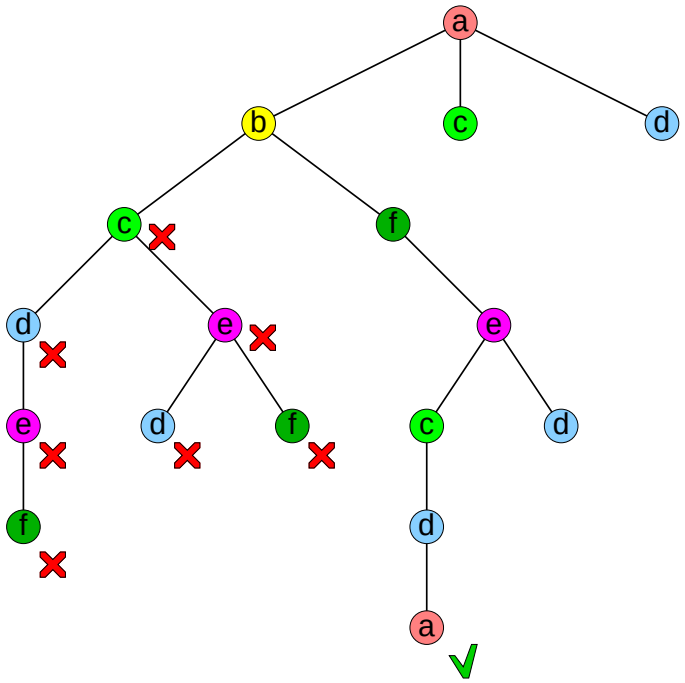
Vind een Hamiltonkring in een gegeven ongerichte graaf.

Voorbeeld: a b f e c d is een Hamiltonkring in nevenstaande graaf, echter a b c d e f is geen Hamiltonkring.



Backtracking: genereer de mogelijke Hamiltonkringen door stap voor stap de knopen te kiezen en controleer tijdens de constructie of de deelpermutatie nog wel een pad/kring voorstelt (de restrictie).

De werking kunnen we beschrijven met de volgende state space tree:



. breid het pad telkens met één knoop uit (keuzes in alfabetisch volgorde)

. in de knopen met een rood kruis backtrackt het algoritme zodra blijkt dat die niet tot een oplossing leiden

Merk op: deze boom beschrijft de werking, maar hoeft niet daadwerkelijk te worden opgebouwd. Alleen het pad corresponderend met de (deel)oplossing die 'nu' wordt uitgebreid wordt bijgehouden.

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.

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 (resp. bovengrens) 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 (**)

(*) goede onder/bovengrens voor snoeien zou bij backtracking ook kunnen, (**) maar dit niet of nauwelijks

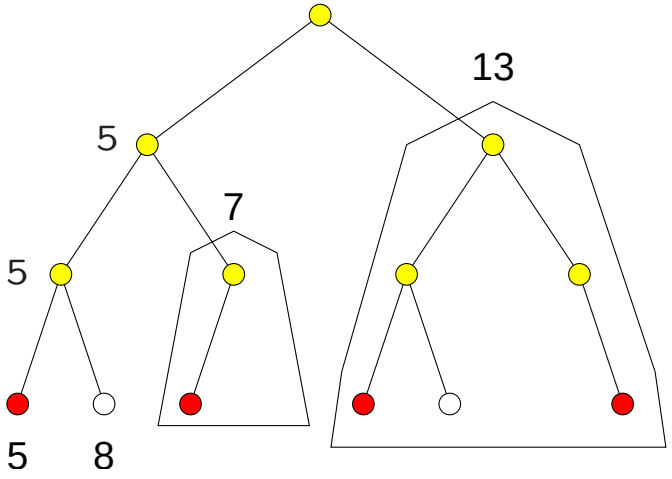
Een branch and bound algoritme breidt een knoop (deeloplossing) niet verder uit als

- de waarde van de ondergrens (bovengrens) bij die knoop niet beter is dan de waarde van de tot dusver gevonden beste oplossing
- de deeloplossing niet meer voldoet aan de restricties (of niet meer uit te breiden is tot een toelaatbare oplossing)
- er een volledige, toelaatbare oplossing gevonden is (dus geen verdere uitbreiding meer mogelijk is): update de beste oplossing

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

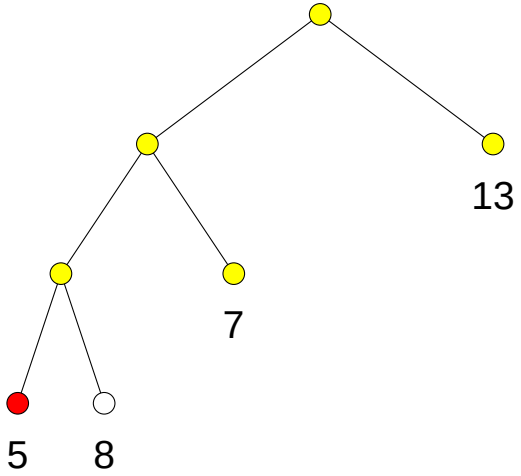
- er worden a.h.w. meerdere deeloplossingen tegelijk bijgehouden, dit in tegenstelling tot backtracking
- in elke stap wordt een van al deze deeloplossingen gekozen, en daarvan worden alle 1-staps-uitbreidingen (kinderen in de state space tree) bekeken en geëvalueerd (d.w.z. ondergrens/bovengrens bepaald)
- zinloze uitbreidingen worden meteen verworpen
- meestal wordt de deeloplossing gekozen die het meest veelbelovend lijkt: best-fit-first

Branch & bound snoeien



state space tree

witte knopen corresponderen met feasible solutions; rode knopen met niet-feasible solutions; de waarden bij de bladeren geven de waarde van de objectfunctie van de bijbehorende oplossing



gesnoeide boom

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

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: $2+3+1+4=10$

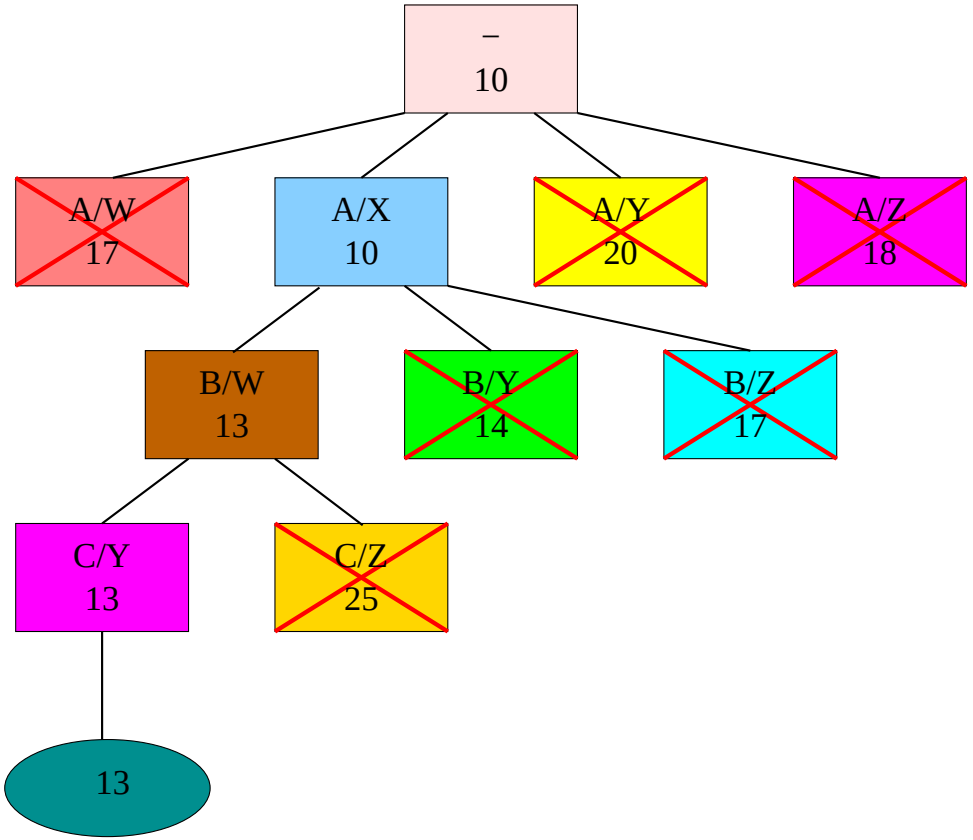
(b) neem uit elke kolom de kleinste waarde en tel die bij elkaar op: $5+2+1+4=12$

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 object-functie door uit elke verdere rij de kleinste waarde van de nog beschikbare jobs te nemen en deze op te tellen. 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.

B&B state space tree



	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 (*).

Los het toewijzingsprobleem op voor onderstaand voorbeeld met behulp van

1. backtracking
2. branch and bound

en vergelijk de hoeveelheid snoeiwerk bij beide methoden.

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

- **Volgende werkcolleges:**

  donderdag 7 mei 2009 in zaal 174
  donderdag 14 mei 2009 in zaal 302/304:
  programmeeropdracht 3
  vrijdag 15 mei 2009 (9.15–10.45) in zaal 174

- **Opgaven:**

  http://www.liacs.nl/home/graaf/ALGO/

- **Volgende colleges:**

  vrijdag 8 mei 2009
  vrijdag 15 mei 2009