- Ongelman edellä aiheuttaa saman muuttujanimen käyttö
- Toisaalta eri lauseiden muuttujanimillä ei ole merkitystä (ennen sidontaa) ja muuttujien nimentä voidaan standardoida
- Mahdollisia samaistuksia on useita, mikä niistä tulisi palauttaa?
- Samaistusalgoritmin edellytetään palauttavan kaikkein yleisimmän samaistuksen
- Mitä vähemmän rajoituksia (sitomisia vakioihin) muuttujasijoitus asettaa, sitä yleisempi se on

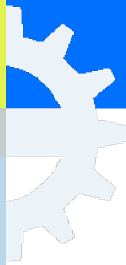
Unify(Tuntee(Jussi, x), Tuntee(y, z))
{ y/Jussi, z/x }
{ y/Jussi, x/Jussi, z/Jussi }

Eteenpäin ketjutus

- Kuten edellä, tarkastelemme Hornin normaalimuodossa olevia tietämyskantoja
- Jokainen lause on siis atomilause tai implikaatio, jonka runko on positiivisten atomilauseiden konjunktio ja kärki on yksittäinen positiivinen atomilause

Teekkari(Jussi)
Ahkera(x)
Teekkari(y) ∧ Ahkera(y) ⇒ DI_2009(y)

- Nyt vaan atomilauseet voivat sisältää myös muuttujia
- Muuttujat ovat aina universaalisesti kvantifioituja

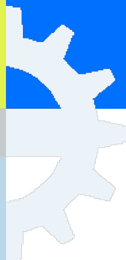


- Kuten propositiologiikassa lähtien liikkeelle faktoista, soveltaen yleistettyä Modus Ponensia voidaan tehdä eteenpäin ketjuttavaa päättelyä
- Nyt on huolehdittava siitä, ettei "uusi" fakta ole pelkkä tunnetun faktan uudelleennimentä

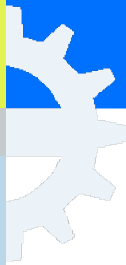
Pitää(x, Karkki)
Pitää(y, Karkki)

- Yleistetyn Modus Ponensin sovelluksena eteenpäin ketjutus on eheä päättelyalgoritmi
- Se on myös täydellinen siinä mielessä, että tietämyskannan mitä tahansa loogista seurausta koskevaan kyselyyn saadaan vastaus

Datalog



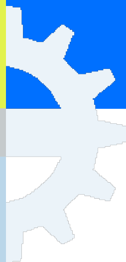
- Datalog tietämyskannassa ei esiinny laisinkaan funktiosymboleja
- Tässä tapauksessa voimme helposti todistaa päättelyn täydellisyyden
- Olkoot tietämyskannan
  - predikaattien lkm p,
  - predikaattien maksimi ariteetti (= argumenttien lkm) k ja
  - vakioiden lkm n
- Vakioarvoihin sidottuja faktoja on korkeintaan pn^k
- Täten näin monen kierroksen jälkeen on saavutettu *kiintopiste* (fixed point), jossa uudet päättelyt eivät enää ole mahdollisia



- Datalogilla polynominen askelten lukumäärä riittää kaikkien loogisten seurausten generoimiseen
- Yleisessä tapauksessa joudumme vetoamaan Herbrandin tulokseen
- Kyselyllä, joka ei ole teorian (= tietämyskannan) looginen seuraus, eteenpäin ketjutus ei välttämättä pysähdy
- Esim. Peanon aksioomilla tietämyskantaan lisättäisiin faktat

LL(S(0)).
LL(S(S(0))).
LL(S(S(S(0)))).
...

- Päättely on vain osittain ratkeavaa



Taaksepäin ketjutus

- Predikaattilogiikassa taaksepäin ketjutuksessa tutkitaan niiden sääntöjen runkoja, joiden kärki samaistuu maalin kanssa
- Rungon kaikista konjunkteista tehdään rekursiivisesti maaleja
- Kun maali samaistuu faktaan – lause, jolla on kärki vaan ei runkoa – ei uusia (ali)maaleja tarvitse generoida
- Syvyyssuuntainen haku
- Palautettava muuttujasijoitus koostetaan kaikkien välivaiheiden ratkaisemiseksi tarvituista sijoituksista
- Prologin päättely perustuu taaksepäin ketjutukseen

Logiikkaohjelmointi

- Prolog, Alain Colmerauer 1972
- Ohjelma = Hornin lauseina ilmaistu tietämyskanta
- Tietämyskantaan kohdistuvat kyselyt
- Suljetun maailman oletus: $\neg\varphi$ oletetaan todeksi, jos lausetta φ ei voi johtaa tietämyskannasta
- Syntaksi:
  - isot kirjaimet muuttujia,
  - pienet vakioita,
  - säännön kärki edeltää runkoa,
  - implikaatiomerkin sijaan `:-`,
  - pilkku konjunktiomerkin tilalla,
  - piste päättää lauseen

```
di_2009(X):- teekkari(X), ahkera(X).
```

- Prologissa on paljon erikoismerkintöjä listoille ja aritmetiikalle

- Ohjelma `append(X,Y,Z)` onnistuu, jos lista `Z` on listojen `X` ja `Y` yhdistämisen (katenoinnin) tulos

```
append([ ],Y,Y).
append([A|X],Y,[A|Z]):- append(X,Y,Z).
```

- Kysely: `append([1],[2],Z)?`

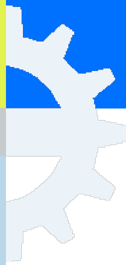
```
Z=[1,2]
```

- Voimme esittää ohjelmalle myös kyselyn

  `append(A,B,[1,2])?`:

  Minkä kahden listan yhdistäminen tuottaa listan `[1,2]`?
- Vastauksena saamme mahdolliset muuttujasidonnat

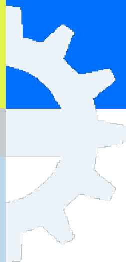
```
A=[ ]       B=[1,2]
A=[1]       B=[2]
A=[1,2]     B=[ ]
```



- Ohjelman lauseet suoritetaan järjestyksessä
- Myös säännön rungon konjunkteja käsitellään järjestyksessä (vasemmalta oikealle)
- Aritmetiikkaa varten on valmiita funktioita, joita ei enää tarvitse päätellä
  - Esim. `X is 4+3` → `X=7`
- Mm. I/O hoituu sisäänrakennetuilla predikaateilla, joilla on sivuvaikutuksia
- Suljetun maailman oletus: negaatio

```
alive(X):- not dead(X).
```

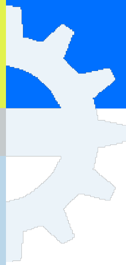
"Kaikki, joiden ei voi todistaa kuolleen, ovat elossa"



- Prologin negaatio ei vastaa logiikan negaatiota (suljetun maailman oletuksella)

```
vapaa_teekkari(X):-
        not naimisissa(X), teekkari(X).
teekkari(pekka).
naimisissa(jussi).
```

- Suljetun maailman oletuksen perusteella `X=pekka` on ohjelman ratkaisu
- Ohjelman suoritus kuitenkin epäonnistuu, koska kun `X=jussi` epäonnistuu rungon ensimmäinen predikaatti
- Jos säännön konjunktit kirjoitettaisiin toiseen järjestykseen, niin se onnistuisi

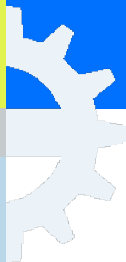


- Yhtäsuuruusvertailu onnistuu, jos tarkasteltavat termit voidaan samaistaa
  - Esim. `X+Y=2+3` → `X=2, Y=3`
- Prolog ei tee kaikkia tarpeellisia tarkistuksia muuttujasijoituksien yhteydessä → Päättely ei ole eheää
- Yleensä ei kuitenkaan ongelmia
- Syvyyssuuntaisen päättelyn seurauksena voi olla ikuinen silmukka

```
path(X,Z):- path(X,Y), link(Y,Z).
path(X,Z):- link(X,Z).
```

- Huolellisuus kuitenkin usein auttaa

```
path(X,Z):- link(X,Z).
path(X,Z):- path(X,Y), link(Y,Z).
```

- Anonyymi muuttuja _

```
member(X,[X|_]).
member(X,[_|Y]):- member(X,Y).
```

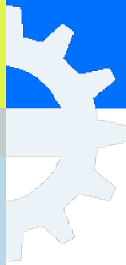
- Toimii sinänsä mainiosti

```
member(d,[a,b,c,d,e,f,g])?
yes
member(2,[3,a,4,f])?
no
```

- Mutta

```
member(a,X)?
member(a,[a,b,r,a,c,a,d,a,b,r,a])?
```

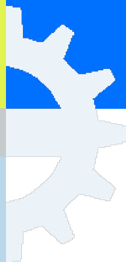
eivät välttämättä toimi toivotulla tavalla



- Prologin suoritusta voidaan karsia *leikkauksella* (cut)
- Negaatio leikkauksen avulla

```
not X:- X, !, fail.
not X.
```

- Edellä `fail` aiheuttaa epäonnistumisen
- Leikkauksen kohdalla kiinnitetään kaikki ne sidonnat, jotka on tehty säännön tutkimisen aloittamisesta lähtien
- Rungon konjunkteille, jotka edeltävät leikkausta, ei enää yritetä muodostaa uusia ratkaisuja
- Muita saman kärjen omaavia sääntöjä ei enää kokeilla



- Prolog voi päätyä samaan ratkaisuun monen päättelypolun kautta
- Tällöin sama ratkaisu palautetaan useita kertoja

```
minimum(X,Y,X):- X<=Y.
minimum(X,Y,Y):- X>=Y.
```

- Molempien sääntöjen kautta löytyyy sama ratkaisu kyselylle `minimum(2,2,M)?`
- Leikkauksen käytössä päättelyn optimointiin on oltava huolellinen

```
minimum(X,Y,X):- X<=Y, !.
minimum(X,Y,Y).
```

- Yo. ohjelma on virheellinen, esimerkiksi `minimum(2,8,8)` on siinä tosi

- Prologin ja logiikkaohjelmoinnin kynnyskysymys tietysti on tehokkuus
- Prologissa on monia tehostuskeinoja käytössä
- Esim. sen sijaan, että alimaalille tuotettaisiin kaikki mahdolliset ratkaisut ennen seuraavaan siirtymistä, Prolog-tulkki tyytyy (toistaiseksi) yhteen
- Samoin muuttujasidonta on kullakin hetkellä yksikäitteinen, vasta umpikujaan ajautuminen ja *peruutus* voi johtaa muuttujan uudelleensitomiseen
- Peruutus edellyttää historia*jäljen* (trail) ylläpitämistä

Resoluutio

- Kurt Gödelin täydellisyyslause (1930) 1. kertaluvun logiikalle: jokaisella loogisella seurauksella on äärellinen todistus
  $T \models \varphi \Leftrightarrow T \vdash \varphi$
- Vasta Robinsonin (1965) resoluutioalgoritmi antoi käytännöllisen menetelmän todistusten muodostamiseksi
- Gödelin tuloksista tunnetumpi on tietysti epätäydellisyyslause: matemaattisen induktion omaavan teorian kaikkia loogisia seurauksia ei voida todistaa aksioomista
- Erityisesti tämä koskee lukuteoriaa, jota siis ei voi aksiomatisoida

- Resoluutiota varten lauseet on muutettava konjunktiiviseen normaalimuotoon.
- Esimerkiksi

  $\forall$ x: [$\forall$ y: Eläin(y) ⇨ Rakastaa(x, y)] ⇨ [$\exists$ y: Rakastaa(y, x)].

- **Implikaation poisto**

  $\forall$ x: [$\neg\forall$ y: $\neg$ Eläin(y) $\vee$ Rakastaa(x, y)] $\vee$ [$\exists$ y: Rakastaa(y, x)].
- **Negaation siirto sisään**

  $\forall$ x: [$\exists$ y: Eläin(y) $\wedge$ $\neg$Rakastaa(x, y)] $\vee$ [$\exists$ y: Rakastaa(y, x)].
- **Muuttujanimien standardointi**

  $\forall$ x: [$\exists$ y: Eläin(y) $\wedge$ $\neg$Rakastaa(x,y)] $\vee$ [$\exists$ z: Rakastaa(z, x)].

- **Skolemisointi**

  $\forall$ x: [Eläin(F(x)) $\wedge$ $\neg$Rakastaa(x, F(x))] $\vee$ Rakastaa(G(x), x).
- **Universaalikvanttorin poisto**

  [Eläin(F(x)) $\wedge$ $\neg$Rakastaa(x, F(x))] $\vee$ Rakastaa(G(x), x).
- **Distributiivisuuden soveltaminen**

  [Eläin(F(x)) $\vee$ Rakastaa(G(x), x)] $\wedge$
  [$\neg$Rakastaa(x, F(x)) $\vee$ Rakastaa(G(x), x)].

- Lopputulos ei enää ole helposti ymmärrettävä, mutta se ei haittaa, koska muunnosproseduuri on automatisoitavissa

- 1. kertaluvun logiikan kaksi literaalia ovat komplementaarisia, jos yksi voidaan samaistaa toisen negaation kanssa
- Täten resoluutioaskel yhdellä komplementaarisella literaalilla on

$$\frac{\ell_1 \vee \ldots \vee \ell_k, \quad m}{(\ell_1 \vee \ldots \vee \ell_{i-1} \vee \ell_{i+1} \vee \ldots \vee \ell_k)(\theta)}$$

missä $\text{Unify}(\ell_i, \neg m) = \theta$

- Resoluutio on myös predikaattilogiikalle täydellinen päättelyjärjestelmä siinä mielessä, että sen avulla voidaan tarkastaa tietämyskannan loogiset seuraukset
- $TK \models \alpha$ todistetaan osoittamalla, että $TK \wedge \neg \alpha$ on toteutumaton ristiriidan avulla

- *Teoreemantodistimissa* logiikkaohjelmoinnin Hornin lauseisiin rajoitettu kieli on laajennettu täyteen 1. kertaluvun logiikkaan
- Myös esim. Prologin tapa antaa kontrollin sekoittaa lauseiden logiikkaa on yleensä poistettu
- Päättely ei riipu lauseiden tai niiden osien kirjoitusjärjestyksestä
- Sovelluskohde: ohjelmien ja laitteiston verifiointi
- Matematiikassa teoreemantodistimet ovat nousseet näkyvään asemaan
- Esimerkiksi 1996 Otter-ohjelman seuraaja ensimmäisenä todisti, että Robbinsin 1933 esittämät aksioomat määrittävät Boolen algebran

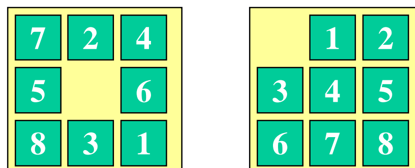
3. ONGELMANRATKAISU JA HAKUALGORITMIT

- Tavoitehakuinen agentti pyrkii ratkaisemaan ongelmia suorittamalla toimintoja, jotka johtavat haluttuihin tiloihin
- Tarkastelemme ensin tilannetta, jossa agentilla ei ole mitään ylimääräistä tietoa ongelmasta
- Sokeassa haussa vain ratkaistavan ongelman rakenne on määritelty
- Agentti pyrkii saavuttamaan maalitilan
- Maailma on staattinen, diskreetti ja deterministinen

- Maailman mahdolliset tilat määräävät etsintäongelman *tila-avaruuden* Σ

- Lähtötilanteessa maailma on *alkutilassa* $s_1 \in \Sigma$
- Agentin tavoitteena on päätyä alkutilasta yhteen maalitiloista $G \subseteq \Sigma$
- Agentin mahdolliset toiminnot määritellään usein *seuraajafunktion* $S: \Sigma \rightarrow \mathcal{P}(\Sigma)$ avulla
- Kullakin tilalla $s \in \Sigma$ on joukko seuraajatiloja $S(s)$, joihin voidaan siirtyä yhdessä askelessa
- Poluilla on ei-negatiivinen *kustannus*, joka yleensä on askelkustannusten summa

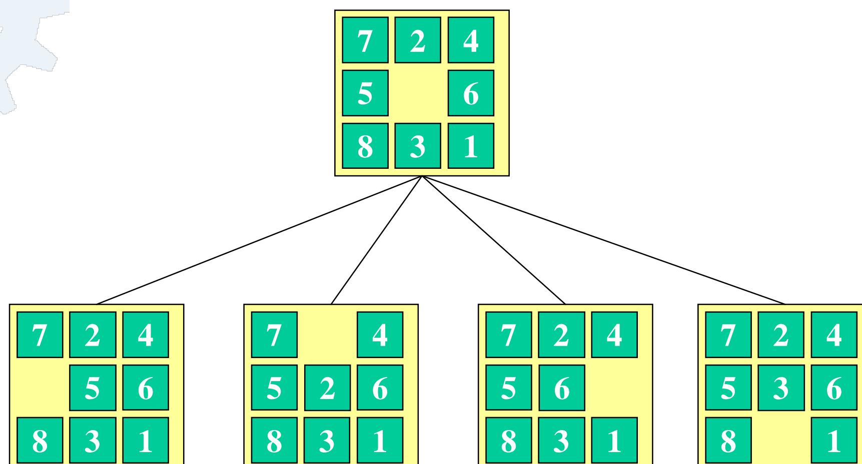
- Edelliset määritelmät määräävät luontevasti suunnatun, painotetun verkon
- Yksinkertaisin tavoite etsinnässä on selvittää onko alkutilasta s_1 saavutettavissa yhtään maalitilaa
- Ongelman varsinainen *ratkaisu* kuitenkin on polku alkutilasta maalitilaan
- Usein tietysti huomioidaan myös polkujen kustannukset ja pyritään löytämään niiden suhteen optimaalinen polku maalitilaan
- Useat tavoitehakuisen agentin tehtävät on helppo formuloida suoraan tässä esityksessä

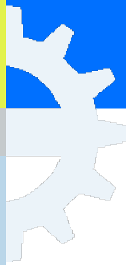


- Esimerkiksi 8-puzzlen maailman mahdollisia tiloja ovat kaikki 9!/2 = 181 440 saavutettavissa olevaa palojen asettelua
- Alkutila on yksi mahdollisista tiloista
- Maalitila on yllä oikealla annettu asetelma
- Seuraajafunktion mahdollisia arvoja ovat tyhjän ruudun siirtäminen vasemmalle, oikealle, ylös tai alas
- Kukin siirto maksaa yhden yksikön ja polun kustannus on siirtojen lukumäärä

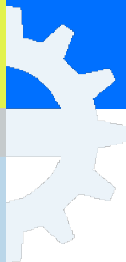
Hakupuu

- Kun maalin etsintä lähtee liikkeelle alkutilasta edeten seuraajafunktion määräämin askelin, niin etsinnän etenemistä voidaan tarkastella puurakenteessa
- Kun hakupuun alkutilaa vastaava juurisolmu laajennetaan, niin siihen sovelletaan seuraajafunktiota, jonka tuloksena hakupuun uusiksi solmuiksi — juuren lapsiksi — luodaan seuraajatiloja vastaavat solmut
- Edelleen muita solmuja laajentamalla voidaan etsintää jatkaa
- Etsintästrategia (hakualgoritmi) määrää missä järjestyksessä solmuja puussa laajennetaan





- Hakupuun solmuihin liitetään
  - tieto sitä vastaavasta tilasta,
  - linkki isäsolmuun,
  - tieto sovelletusta toiminnosta, joka on johtanut solmun laajentamiseen,
  - polun kustannus alkutilasta lähtien tätä solmua vastaavaan tilaan saakka ja
  - solmun syvyys
- Hakupuun globaaleja ominaisuuksia ovat
  - (keskimääräinen tai maksimaalinen) haarautumisaste b,
  - matalimmalla olevan maalisolmun syvyys d sekä
  - tila-avaruuden pisimmän polun pituus m



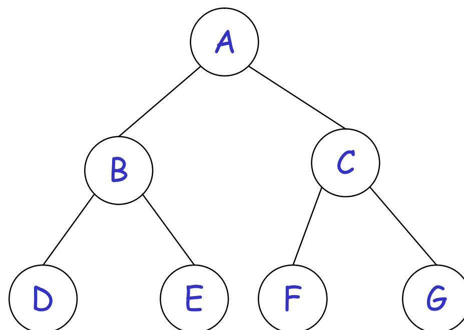
- Hakualgoritmit toteutetaan normaalin puun läpikäynnin erikoistapauksina
- Haun aikavaativuutta mitataan useimmiten puuhun luotujen solmujen lukumäärällä
- Tilavaativuus puolestaan mittaa yhtaikaisesti muistissa säilytettävien solmujen lukumäärää
- Hakualgoritmi on *täydellinen*, jos jokin maalitila voidaan saavuttaa alkutilasta
- Täydellisen algoritmin palauttama ratkaisu ei välttämättä ole *optimaalinen*: alkutilasta voi olla saavutettavissa useita eri kustannuksen omaavia maaleja

Leveyssuuntainen haku

- Kun puun läpikäyntijärjestys on esijärjestys, niin hakupuun solmut tulevat käsitellyiksi taso kerrallaan
- Kaikki samalla syvyydellä olevat solmut laajennetaan ennen kuin yhtään alemman tason solmuista käsitellään
- Jos maalitila on syvyydellä d, niin kaikki edeltävien d-1 tason solmut on käsiteltävä (ja talletettava) ennen sen löytymistä
- Kun hakupuun haarautumisaste on b, niin aika- ja tilavaativuus on pahimmillaan

$$b + b^2 + \ldots + b^d + (b^{d+1} - b) = O(b^{d+1})$$

- Leveyssuuntaisen haun heikkous yleisesti ottaen on sen syvyyden suhteen eksponentiaalinen ajan ja tilan käyttö
- Erityisesti tarve säilyttää kaikki jo läpikäydyt tilat aiheuttaa ongelmia



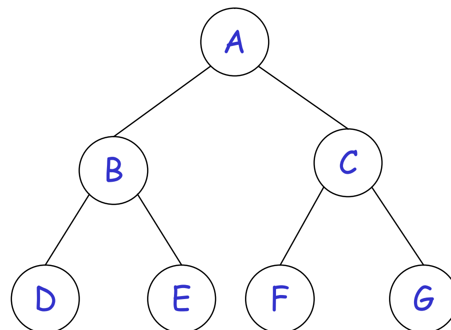
- Esim. kun puun haarautumisaste on 10, solmuja läpikäydään 10 000 sekunnissa ja solmujen vaatima talletustila on 1 000 tavua, niin

Syvyys	Solmuja	Aika	Muistitila
2	1 100	.11 s	1 MB (10^6)
4	111 100	11 s	106 MB
6	10^7	19 min	10 GB (10^9)
8	10^9	31 h	1 teraB (10^{12})
10	10^{11}	129 vrk	101 teraB
12	10^{13}	35 a	10 petaB (10^{15})
14	10^{15}	3 523 a	1 exaB (10^{18})

- Leveyssuuntainen haku palauttaa optimaalisen ratkaisun, jos kaikkien askelten kustannus on sama
- Toisaalta tähän erikoistapaukseen voidaan soveltaa *ahnetta* (greedy) algoritmia, joka aina laajentaa sen solmun, jonka tähänastinen polun kustannus on pienin
- Jos ongelman kaikki askelkustannukset ovat aidosti positiivisia, niin ahneen algoritmin palauttama ratkaisu on optimaalinen
- Algoritmin tilavaativuus on edelleenkin suuri
- Kun askelkustannukset ovat kaikki samoja, on ahne algoritmi yhtenevainen leveyssuuntaisen haun kanssa

Syvyyssuuntainen haku

- Kun hakupuun solmut laajennetaan sisäjärjestyksessä, niin hakualgoritmi käsittelee aina ensinnä syvimmällä olevan solmun
- Kun yksi puun haara juuresta lehteen on tullut käsitellyksi, haku peruuttaa puussa takaisin syvimmällä olevaan käsittelemättömään solmuun
- Syvyyssuuntaisen haun hyvä puoli on pieni muistitilavaatimus
- Kullakin hetkellä riittää pitää muistissa yksi polku juuresta lehteen ja polulle kuuluvien solmujen laajentamattomat sisarsolmut
- Tilavaativuus on siis $bm+1$



- Edellisen esimerkin tapauksessa, jossa maalisolmu on tasolla 12 muistia vaaditaan vain 118 kB
- Jos hakupuu on ääretön, niin syvyyssuuntainen haku ei ole täydellinen menetelmä
- Ainut maalisolmu voi sijaita vasta viimeksi tutkittavassa puun haarassa
- Pahimmassa tapauksessa syvyyssuuntainen hakukin vaatii eksponentiaalisen määrän aikaa: $O(b^m)$
- Pahimmillaan $m \gg d$, syvyyssuuntaisen haun vaatima aika voi olla paljon suurempi kuin leveyssuuntaisen haun
- Myöskään löydetyn ratkaisun optimaalisuutta ei voida taata

Rajoitetun syvyyden haku

- Äärettömän syvyisen haaran tutkiminen voidaan estää rajoittamalla etsintä syvyyteen ℓ
- Tason ℓ solmuja käsitellään lehtinä

- Syvyyssuuntainen haku on erikoistapaus, jossa $\ell = m$
- Tapauksessa $d \leq \ell$ hakualgoritmi on täydellinen, mutta yleisesti ratkaisun löytymistä ei voi taata

- Algoritmi ei myöskään (tietysti) takaa optimaalisen ratkaisun löytämistä
- Aikavaativuus on nyt $O(b^\ell)$ ja tilavaativuus $O(b\ell)$

Iteratiivinen syventäminen

- Rajoitetun syvyyden haun ja yleisen syvyyssuuntaisen haun hyvät puolet voidaan yhdistää, kun annetaan parametrin ℓ arvon vähitellen kasvaa
- Esim. $\ell = 0, 1, 2, \ldots$ kunnes maalisolmu löytyy

- Itseasiassa näin saadaan yhdistettyä myös leveys- ja syvyyssuuntaisen haun hyvät puolet
- Tilavaativuus pysyy kurissa, koska hakualgoritmina on syvyyssuuntainen haku

- Toisaalta parametrin ℓ vähittäinen kasvattaminen takaa, että menetelmä on täydellinen ja optimaalinen silloin kun polun kustannus on solmun syvyyden suhteen ei-vähenevä

Kaksisuuntainen haku

- Jos edeltäjäsolmut ovat helposti laskettavissa — esim. $Pred(n) = S(n)$ — niin haku voi edetä yht'aikaisesti alkutilasta eteenpäin ja maalitilasta taaksepäin
- Etsintä päättyy kun haut kohtaavat
- Motivaatio tälle ajatukselle on, että $2b^{d/2} \ll b^d$
- Jos molempiin suuntiin etenevät haut toteutetaan leveyssuuntaisella haulla, on menetelmä täydellinen ja johtaa optimaaliseen tulokseen
- Jos maalitila on yksikäsitteinen, niin taaksepäin eteneminen on selvää, mutta monet maalitilat voivat edellyttää uusien väliaikaisten maalitilojen luomista