Föreläsning 8: Problemträd

- Problemträd
- Breddenförstsökning
- Rekursiv djupetförst
- Djupetförst med stack
- Grafer

Problemträd

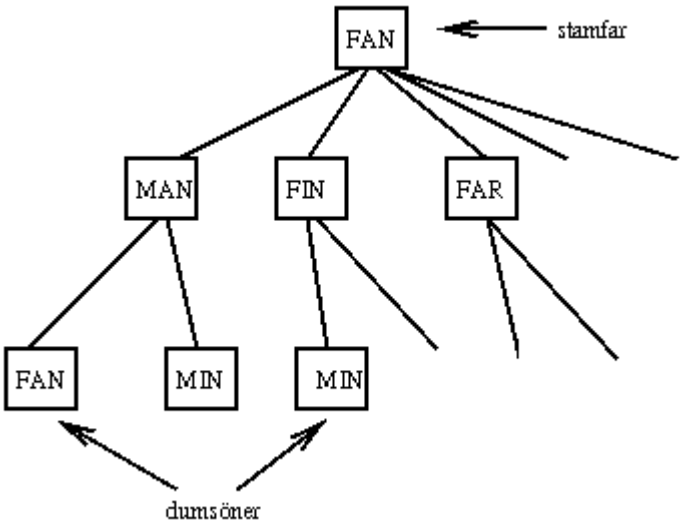
En mycket stor klass av praktiska problem kan beskrivas med problemträd och lösas med trädgenomgång, bredden först eller djupet först. Laboration 4 går ut på att finna kortaste vägen från **fan** till **gud** genom att byta en bokstav i taget och bara använda ord i ordlistan, till exempel så här:

fan -> man -> mun -> tun -> tur -> hur -> hud -> gud

Problemträd uppkommer ständigt i praktiken. Man brukar kalla startobjektet för *stamfar* och objekten under det för *söner*.

Breddenförstsökning

Problemträdets stamfar **fan** har sönerna **fin**, **man**, **far** med flera, sonsönerna **hin**, **mun**, **får** osv. Enligt kedjan ovan är **gud** sonsonsonsonsonsonson till **fan**, men gud finns säkert redan tidigare i problemträdet. För att finna den första förekomsten gör man en breddenförstsökning enligt följande.



Lägg stamfadern som första och enda post i en kö. Gör sedan följande om och om igen: Plocka ut den första ur kön, skapa alla söner till denne och lägg in dom sist i kön. Första förekomsten av det sökta ordet ger kortaste lösningen.

Man kan spara in både tid och utrymme om man undviker att skapa barn som är kopior av tidigare släktingar (t ex **man**s son **fan**), som vi kan kalla *dumsöner (eller dumdöttrar).*

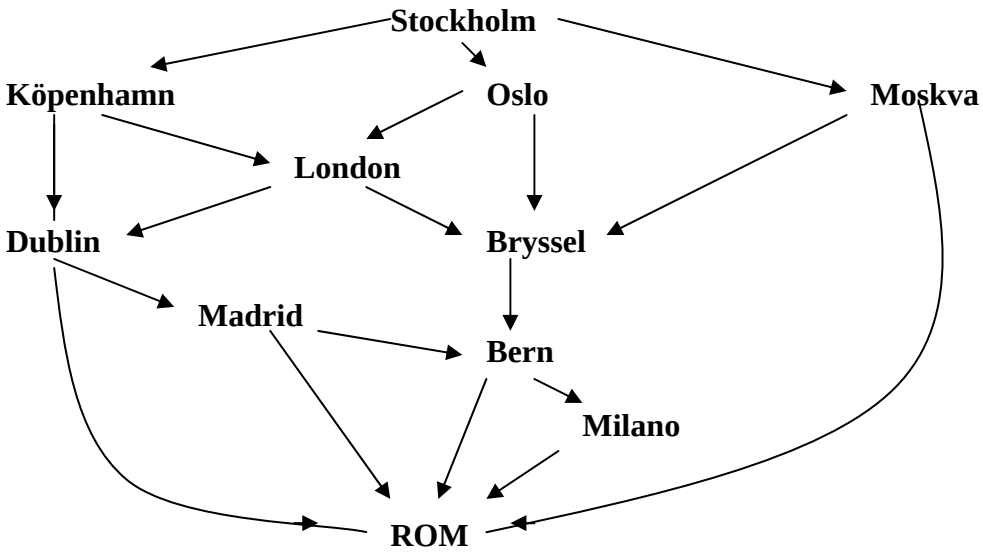
Breddenförstsökningsalgoritmen kan sammanfattas så här.

1. Lägg stamfadern i kön.
2. Ta ut den första posten ur kön.
3. Skapa alla dess söner och lägg in dom i kön.
4. Om någon av sönerna är lösningen så är vi klara. Annars,
   - upprepa från punkt 2.
5. När lösningen hittas, följ faderspekarna och skriv ut kedjan.

Om man bara lägger själva orden i kön finns det ingen möjlighet att i efterhand tala om vägen från fan till gud. Därför bör man för varje nytt ord skapa en liten post som innehåller ordet och en referens till fadern. Breddenförstsökning ger alltid den *kortaste* lösningen. Ofta är det den man är ute efter. Några andra problemexempel är följande.

Flygresa från Stockholm till Rom

Stockholm är stamfar, destinationer med direktflyg från Stockholm blir söner och så vidare. Breddenförstsökning ger en resa med så få mellanlandningar som möjligt.



Dynamisk programmering

Att det är minst antal mellanlandningar behöver inte betyda att det är den resrutt som tar kortast tid. Att gå igenom alla kombinationer av resrutter för att hitta den bästa resrutten är inte så bra. Istället kan använda sig av *dynamisk programmering* som i innebär att man sparar undan och utnyttjar tidigare uträknade värden när man ska beräkna nästa värde.

Gör så här: Börja från slutet d.v.s ROM. För varje stad, fyll i sammanlagd restid och varifrån du rest. Om du stöter på en stad du redan behandlat, jämför och spara den bästa restiden. Metoden förutsätter att alla resrutter är enkelriktade mot resmålet d.v.s inga returresor finns med.

Rekursiv djupetförstsökning

Djupetförstsökning skiljer sig från breddenförstsökning i två avseenden:

- Den första lösning man hittar är inte nödvändigtvis den kortaste.
- Metoden fungerar inte om problemträdet har oändligt djup till skillnad från breddenförst som kommer att stanna vid första lösningen.

Ett exempel är *åttadamersproblemet* som innebär att man ska placera åtta damer på ett schackbräde så att ingen dam står på samma vågräta, lodräta eller diagonala linje som någon annan. Problemträdets stamfar är ett tomt bräde. Dom åtta sönerna har en dam placerad på översta raden, sonsönerna ytterligare en dam på näst översta raden etc.

Den första idén man får är ju att representera schackbrädet med en matris. Men lösningen blir enklare om man använder en vektor, där varje vektorelement är ett heltal som representerar damens position på just den raden.
`queenPos[1]=5;` betyder då att damen på första raden står i position 5.

Rekursiv tanke:
För att placera ut en dam på rad `row`:

- Prova att placera damen i varje position på raden i tur och ordning.
- Kan damen stå i den positionen så placeras nästa dam på rad `row+1` ...
- ... om inte `row=8` för då har man hittat en lösning.

```
class Queens:
    queens_position = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

    def position_is_OK(self, row) :
        previous_rows = range(1, row)
        previous_rows.reverse()
        for i in previous_rows:
            if self.queens_position[i] == self.queens_position[row]:
                return False
            if abs(self.queens_position[i] -      # testa diagonal
                   self.queens_position[row]) == abs(i-row):
                return False
        return True

    def test_row(self, row):
        for column in xrange(1, 9):
            self.queens_position[row] = column
            if self.position_is_OK(row):
                if row == 8 :
                    print self.queens_position
                else :
                    self.test_row(row + 1)

q = Queens()
q.test_row(1)
```

För enkelhets skull är schackraderna här numrerade 1..8 `(queen_position[0]` används inte`)`.

Djupetförstsökning med stack

Djupetförstsökning kan också programmeras som breddenförstsökningen, med den lilla skillnaden att kön byts mot en **stack**. Här följer några fler exempel på problem som kan lösas med djupetförstsökning.

Hitta ut ur labyrint

En välkänd praktisk metod att utforska en labyrint, uppfunnen av den förhistoriska datalogen Ariadne, är att ha ett garnnystan med ena änden fastknuten i startpunkten. Man går så långt man kan, markerar med krita var man varit, går bara outforskade vägar framåt och backar en bit längs snöret när man kör fast.

Problemträdet har startpositionen som stamfar, alla positioner på ett stegs avstånd som söner och så vidare. En position som man varit på förut är en dumson.

Luddes portkodssekvens

En teknolog som glömt sin fyrsiffriga portkod tryckte sej igenom alla tiotusen kombinationer så här.
`00000001000200030004000500060007000800090010011...9999`
Det kräver fyrtiotusen tryckningar. Men man kan klara sej med bara tiotusentre tryckningar om man har en supersmart sekvens där varje fyrsiffrigt tal förekommer någonstans. Hur ser sekvensen ut?

Problemträdets stamfar **0000** har tio söner **0000**, **0001**,..., **0009**, varav den förste är dumson. Breddenförst eller djupetförst? Vi vet att trädet har djupet tiotusen och att alla lösningar är lika långa, därför går djupetförst bra. Men breddenförst skulle kräva biljoner poster!

För att ta gå ner i trädet behöver vi ett spara undan de tal vi redan använt (jfr *dumsonträd)*. Vi kan t.ex. använda oss av en hakvektor med boolska variabler.

Grafer

De problemträd vi tagit upp här är specialfall av *grafer*. En allmänn graf består av en samling hörn (*vertices*) med kanter (*edges*) emellan. Grafer får man läsa mer om i kursen 5B1118 Diskret matematik.