METOD 1. GIRIGA ALGORITMER
(GREEDY ALGORITHMS)

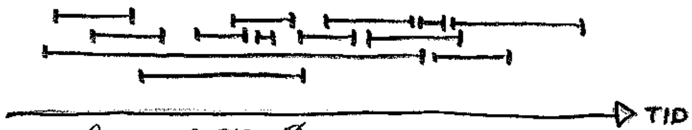
ALGORITMER SOM LÖSER, ITERATIVT, EN BIT AV PROBLEMET I TAGET.
I VARJE STEG GÖRS DET SOM GER BÄST UTDELNING/KOSTAR MINST.

> GIRIGA ALGORITMER ÄR SPECIELLT ANVÄNDBARA FÖR APPROXIMATIONSALGORITMER OCH HEURISTIKER.
> IBLAND KAN DOCK GIRIGA ALGORITMER GE OPTIMALA LÖSNINGAR. DÅ MÅSTE MAN BEVISA ATT ALGORITMEN GÖR DET.

EXEMPEL PÅ PROBLEM SOM KAN LÖSAS OPTIMALT MED EN GIRIG ALGORITM:

- HITTA KORTASTE VÄGEN MELLAN TVÅ PUNKTER I EN GRAF MED KANTVIKTER
- HITTA MINIMALA TRÄDET SOM SPÄNNER UPP EN GRAF
- GIVET EN MÄNGD AKTIVITETER MED START- OCH SLUTTIDER, HITTA STÖRSTA MÄNGDEN AKTIVITETER SOM INTE ÖVERLAPPAR

GIRIG ALGORITM FÖR AKTIVITETSPROBLEMET



IF $n=0$ THEN RETURN $\emptyset$
SORTERA AKTIVITETERNA EFTER SLUTTID f_i
  SÅ ATT $f_1 \le f_2 \le f_3 \le \ldots \le f_n$
$S \leftarrow \{1\}$; current $\leftarrow f_1$
FOR $i \leftarrow 2$ TO n DO
  IF $s_i \ge$ current THEN
    $S \leftarrow S \cup \{i\}$; current $\leftarrow f_i$
RETURN S

BEVIS:

LÅT D VARA EN OPTIMAL DELMÄNGD AKTIVITETER SORTERADE EFTER STIGANDE SLUTTID

OM D ÄR TOM ELLER BESTÅR AV ETT ELEMENT GÖR ALGORITMEN RÄTT.
LÅT k VARA FÖRSTA AKTIVITETEN I D.

OM $k=1$ VÄLJER ALGORITMEN SAMMA AKTIVITET (1)
$k \ne 1$ VÄLJER ALGORITMEN EN AKTIVITET MED SLUTTID $f_1 \le f_k$
VI KAN ALLTSÅ BYTA k MOT 1 I D OCH FORTFARANDE HA EN LÖSNING!

INGEN AKTIVITET MED STARTTID $\le f_1$ (UTÖVER 1) KAN VARA MED I D, SÅ VI PLOCKAR BORT DESSA.

NU HAR VI SAMMA PROBLEM FAST MED EN MINDRE MÄNGD AKTIVITETER (SOM INTE ÖVERLAPPAR MED 1).
INDUKTION $\forall$

METOD 2 TOTALSÖKNING
(EXHAUSTIVE SEARCH)

- GÅ IGENOM ALLA TÄNKBARA LÖSNINGAR OCH KOLLA OM DET ÄR DEN SÖKTA LÖSNINGEN
  DETTA GÖRS MED FÖRDEL REKURSIVT.

OFTA ÄR ANTALET TÄNKBARA LÖSNINGAR EXPONENTIELLT MÅNGA (T EX 2^n) OCH DÅ GÅR DET BARA ATT ANVÄNDA FÖR SMÅ n.
TOTALSÖKNING ÄR EN METOD MAN TAR TILL I SISTA HAND.

DET SVÅRASTE MED TOTALSÖKNING ÄR NORMALT ATT SE TILL ATT MAN GÅR IGENOM VARJE TÄNKBAR LÖSNING EN (OCH HELST INTE MER ÄN EN) GÅNG. ATT SEDAN KOLLA OM DET ÄR DEN SÖKTA (ELLER OPTIMALA) LÖSNINGEN BRUKAR VARA LÄTT.

IBLAND KAN MAN REDAN INNAN EN TÄNKBAR LÖSNING ÄR FÄRDIGKONSTRUERAD SE ATT DEN INTE ÄR MÖJLIG SOM LÖSNING. DÅ KAN MAN STRUNTA I ATT GÅ VIDARE MED DEN OCH ISTÄLLET GÅ TILLBAKA OCH KONSTRUERA NÄSTA MÖJLIGA LÖSNING.
DETTA KALLAS BACKTRACKING.

EXEMPEL: LÖSNING AV DET GENERELLA HANDELSRESANDEPROBLEMET.

TSP – HANDELSRESANDEPROBLEMET
(TRAVELING SALESPERSON PROBLEM)

HITTA KORTASTE TUREN SOM PASSERAR ALLA STÄDER EN GÅNG.

OLIKA VARIANTER AV PROBLEMET:

- GENERELL TSP
  PROBLEMINSTANS: GRAF MED KANTVIKTER
- EUKLIDISK TSP I DIMENSION d
  PROBLEMINSTANS: STÄDERNA GIVNA SOM KOORDINATER I $\mathbb{R}^d$

LÖSNING AV GENERELL TSP MED TOTALSÖKNING

DATASTRUKTURER: PERM[1..n] HÄR KONSTRUERAR VI VARJE TÄNKBAR LÖSNING, DVS PERMUTATION AV STÄDERNA

VISITED[1..n] ANGER OM EN STAD BESÖKTS HITTILLS I DEN PERMUTATION SOM KONSTRUERAS

```
TSP(n, d[1..n,1..n]) =
  minlen ← ∞;
  FOR i ← 1 TO n DO VISITED[i] ← FALSE;
  FOR i ← 1 TO n DO
    PERM[1] ← i; VISITED[i] ← TRUE;
    CheckPerm(2, 0);
    VISITED[i] ← FALSE;
  RETURN minlen

CheckPerm(k, length) =
  IF k>n THEN
    totalLength ← length + d[PERM[n], PERM[1]];
    IF totalLength < minlen THEN minlen ← totalLength;
  ELSE  ← BACKTRACKING KAN INFÖRAS HÄR MED SATSEN
          IF length < minlen THEN
    FOR i ← 1 TO n DO
      IF NOT VISITED[i] THEN
        PERM[k] ← i; VISITED[i] ← TRUE;
        CheckPerm(k+1, length+d[PERM[k-1], i]);
        VISITED[i] ← FALSE;
```

TIDSKOMPLEXITET: $O(n^2 \cdot n!)$ VILKET ENKELT KAN MINSKAS TILL $O(n!)$ OM MAN HÅLLER REDA PÅ VILKA STÄDER SOM ÄNNU INTE BESÖKTS EFFEKTIVARE (T EX MED EN KÖ)

8-DAMSPROBLEMET

PLACERA UT 8 DAMER PÅ ETT SCHACKBRÄDE UTAN ATT NÅGON PJÄS HOTAR NÅGON ANNAN!

TOTALSÖKNINGSALGORITM:

PLACERING AV EN DAM PÅ RAD row:

- PRÖVA ATT PLACERA DAMEN I VARJE POSITION PÅ RADEN I TUR OCH ORDNING
  - KAN DAMEN STÅ OHOTAD I DEN POSITIONEN SÅ PLACERAS NÄSTA DAM PÅ RAD row+1 UT PÅ SAMMA SÄTT, OM INTE row=8 FÖR DÅ HAR MAN HITTAT EN LÖSNING.

```
int queenpos[9];
void TestRow(int row)
{ int i;
  for (i=1; i<=8; i++) {
    queenpos[row] = i;
    if (PosOK(row)) {
      if (row == 8) WriteSolution();
      else TestRow(row+1);
  } }
}
```

STARTA MED TestRow(1);