Einschub: Konstanten

Name : constant Datentyp := Ausdruck, der an dieser Stelle vollständig ausgerechnet werden kann

Beispiele:

```
Pi : constant Float := 3.141592 6;
Piquadrat : constant Float := Pi * Pi;
Sonntag : constant Wochentag := So;
Zwischenraum : constant Character := ' ';
```

Konstanten dürfen nicht im Programm verändert werden. Verboten ist also: Pi := Pi + 1.0

aber auch Pi := Pi oder der Prozeduraufruf PR(Pi), sofern die formale Parameter der Prozeder PR out oder in out ist.

Beispiel Permutationen

Standardproblem: Travelling Sales Person Problem (TSP)

Gegeben sind n Städte und deren Entfernungen

d_{ij} = Entfernung von i nach j $(\in \mathbb{R}^+)$.

Gesucht: "die" kürzeste Rundreise, d.h. die Permutation $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ der n Zahlen mit

$d_\pi \leq d_{\pi'}$ für alle Permutation $\pi' = (\pi'_1, \ldots, \pi'_n)$.

wobei

$$d_\pi = d_{\pi_1 \pi_2} + d_{\pi_2 \pi_3} + d_{\pi_3 \pi_4} + \ldots + d_{\pi_{n-1} \pi_n} + d_{\pi_n \pi_1}$$

die Gesamtentfernung angibt, wenn man die Städte in der Reihenfolge $\pi_1 \to \pi_2 \to \pi_3 \to \ldots \to \pi_n \to \pi_1$ durchläuft.

Lösungsalgorithmus

Berechne für alle Permutationen π den Wert d_π und merke das Minimum.

(Sei hier n = 50.)

```
procedure TSP is    n: constant Integer := 50;
   i, j : Integer ; R, MinR : Float; Ende: Boolean ;
   D: array (1..n, 1..n) of Float; -- Entfernungen
   P: array (1..n) of 1..n ;       -- aktuelle Permutation

   procedure NextP (schluss : in out Boolean) is
      i, j, k: 0.. n ;
   begin      -- berechne die nächste Permutation;
              -- falls dies die Anfangspermutation 123...n
              -- ist, dann setze schluss := true, sonst false
   end;

begin -- lies D ein
   for k in 1..n loop P(k) := k; end loop;
   Ende := false; MinR := "∞";
   while not Ende loop
         NextP(Ende);
         R := D(P(n), P(1));
         for k in 1..n-1 loop
             R := R + D(P(k), P(k+1)); end loop;
         if R < MinR then MinR := R; end if;
   end loop;
   Put (MinR);
end;
```

Wie durchläuft man systematisch <u>alle</u> Permutationen?

Man orientiert sich an den Zahlen, die durch eine Permutation dargestellt werden!

3! = 6 "Zahlen"

1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1

oder

4! = 24 "Zahlen"

1 2 3 4
1 2 4 3
1 3 2 4
1 3 4 2
1 4 2 3
1 4 3 2
2 1 3 4
2 1 4 3
2 3 1 4
2 3 4 1
2 4 1 3
2 4 3 1
3 1 2 4
⋮
4 3 1 2
4 3 2 1

Also: Wie erhält man aus der "Zahl"
$\pi_1 \pi_2 \pi_3 \ldots \pi_n$ die nächstgrößere Zahl, die nur aus den "Ziffern" $1, 2, \ldots, n$ (und jede genau einmal) besteht?

Schritt 1: Bestimme die aufsteigende Kette von hinten und setze die Variable i auf die davorstehende Ziffer:

Position: 1 2 3 4 5 6 7 (⇐ P im Program)

7 2 3 6 5 4 1 gegeben

i ↑ aufsteigende "Kette" (= Folge von Ziffern)

Schritte 2: Bestimme die kleinste Ziffer zwischen den Positionen $(i+1)$ und n, die größer als $P(i)$ ist. Diese stehe an Position j.

i j n
↓ ↓ ↓
7 2 3 6 5 4 1

Schritt 3: Vertausche $P(i)$ und $P(j)$ und drehe die Kette $P(i+1)\, P(i+2) \ldots P(n)$ um:

i j n
↓ ↓ ↓
7 2 4 6 5 3 1 ⇒ 7 2 4 1 3 5 6

umdrehen ("invertieren")

■

```
procedure NextP ( schluss : in out Boolean ) is
   i, j, h, k : 0 .. n ;
begin   i := n - 1;
      while P(i) > P(i+1) loop
            i := i - 1 ; end loop ;
      if i > 0 then  j := n ;
            while P(j) ≤ P(i) loop
                  j := j - 1 ; end loop ;
            h := P(i); P(i) := P(j); P(j) := h;
            i := i + 1 ; k := n ;
            while i < k loop
               h := P(i); P(i) := P(k); P(k) := h ;
               i := i + 1; k := k - 1;
            end loop;
      else schluss := true ; end if ;
end NextP ;
```