Algorithmus

1. Mai 1999

Der Datentyp **TARIFMODELL_ARRAY** enthält alle verfügbaren Tarifmodelle. Mit dem ersten Index k werden die verschiedenen Tarifmodelle durchlaufen, der zweite Index l bestimmt die Ortszonen und i die Zeitzonen. Bei all diesen Operationen wird davon ausgegangen, daß die Tarifmodelle auf einheitliche Orts- und Zeitzonen umgerechnet wurden.

Die erste Funktion bestimmt die Kosten eines Gesprächs (aus dem GESPRÄCH_ARRAY), das gewünschte Array wird über eine Index ausgewählt.

```
/**
/*Berechnet die (relativen) Kosten eines Gesprächs
/*mit einem bestimmten Tarifmodell
*/
FUNCTION call_cost(INT: GesprächNummer,
                   INT: TarifmodellNummer,
                   GESPRÄCH_ARRAY(n,m): cdr,
                   TARIFMODELL_ARRAY(k,l,i): modell)
         RETURN FLOAT IS
BEGIN
  VORWAHL: StartVorwahl, ZielVorwahl;
  INT OrtsZone;
  INT ZeitZone;
  INT Dauer;
  FLOAT Kosten = 0;

  StartVorwahl = cdr(GesprächNummer, 1);
  ZielVorwahl  = cdr(GesprächNummer, 2);
  Dauer = cdr(GesprächNummer, 3);

  /*Ermittlung der kleinsten gemeinsamen Ortszone aus
    Startvorwahl und Zielvorwahl */
  OrtsZone = to_ortszone(StartVorwahl, ZielVorwahl);

  /*Im ARRAY modell sind die normierten Kosten aller
    verfügbaren Tarifmodelle gespeichert. */

Kosten = modell(TarifmodellNummer, OrtsZone, ZeitZone)
               * Dauer;

         RETURN kosten;
```

```
END call_cost;
```

Der Datentyp **GESPRÄCH_ARRAY** enthält eine gesamten CDR (Menge von Gesprächen).

Dies Funktion berechnet nun die niedrigsten Kosten eines gesamten CDRs für eine fest vorgegebene Menge von Tarifmodellen. Auch hier werden die gewünschten Tarifmodell über einen Index ausgewählt.

```
/**
/*Berechnet die niedrigsten Kosten eines CDR
/*für die vorgegeben Menge von Tarifmodellen
*/
FUNCTION cdr_cost(FIXKOSTEN_VECTOR(i): VectorGrundKosten,
                  TARIFMODELL_ARRAY(k,l,i): modell,
                  GESPRÄCH_ARRAY(n,m): cdr,
                  TARIFMODELL_VECTOR(i): tarifmodelle)
         RETURN FLOAT IS
BEGIN
  FLOAT GrundKosten = 0;              //Grundgebühr
  FLOAT GesamtKosten = 0;
  FLOAT GesamtKostenTemp1 = 10E50;
  FLOAT GesamtKostenTemp2 = 10E50;

  /*Die Grundkosten müssen für alle für die Optimierung
    verwendeten Tarifmodelle berücksichtigt werden.
    Die Indizes von FIXKOSTEN_VECTOR(i) und TARIFMODELL_ARRAY
    sind aufeinander abgestimmt, d.h. gleicher Index,
    gleiches Tarifmodell*/
  FOR i IN 1..tarifmodelle'length LOOP
    GrundKosten = GrundKosten +
                  VectorGrundKosten(tarimodelle(i));
  END LOOP;
  GesamtKosten = GrundKosten;

  /*Durchlaufen der einzelnen Gespräche*/
  FOR i IN 1..cdr(1)'length LOOP
    /*Durchlaufen der (berücksichtigten) Tarifmodelle;
      es wird für das konkrete Gespräch der billigste
      Anbieter gesucht*/
     FOR j IN 1..tarifmodelle'length LOOP

GesamtKostenTemp1 = call_cost(i, tarifmodelle(j),
                                  cdr, modell);
       IF GesamtKostenTemp1 < GesamtKostenTemp2 THEN
         GesamtKostenTemp2 = GesamtKostenTemp1;
       END IF;
     END LOOP;
     GesamtKosten = GesamtKosten + GesamtKostenTemp2;
  END LOOP;
```

```
  RETURN GesamtKosten;
END cdr_cost;
```

Bei dieser Funktion wird davon ausgegangen, daß die gesamten Gesprächskosten genau dann minimiert werden, wenn jedes einzelne Gespräch mit dem billigsten Tarifmodell geführt wird. Der Parameter *tarifmodelle* enthält die Indizes der zu berücksichtigenden Tarifmodelle.

```
/**
/*Ermittelt den besten Providermix aus allen Tarifmodellen
*/
FUNCTION best_modell(INT: a) RETURN BESTMODELL_VECTOR IS
BEGIN
  INT: n; //Anzahl aller Tarifmodelle
  FLOAT: kosten = 10E50;
  FLOAT: kostenTemp = 10E50;
  FIXKOSTEN_VECTOR(i): VektorGrundKosten;
  /*Enthält alle zu berücksichtigenden Tarifmodelle*/
  TARIFMODELL_ARRAY(k,l,i): modell;
  GESPRÄCH_ARRAY(n,m): cdr;
  /*dieser Vektor legt fest, welche Tarifmodelle für die
    Optimierung verwendet werden sollen*/
  TARIFMODELL_VECTOR(i,k): tarifmodelle;
  /*Enthält die Indizes des besten Tarifmodellmixes*/
  BESTMODELL_VECTOR(i): best_mix;

  n = modell(1)'length;

  /*Ist a die maximale Anzahl der zu berücksichtigenden
    Tarifmodelle, so müssen alle i-elementigen (1<= i <=a)
    Teilmengen aller Tarifmodelle aus modelle
    berücksichtigt werden*/
  FOR i IN 1..a LOOP

/*getSubSet liefert ein Array, daß alle i-elementigen
      Teilmengen einer n-elementigen Menge enthält;
      auch hier wird mit dem Index der Tarifmodelle
      gearbeitet*/
    tarifmodelle = getSubSets(n,i);
    FOR j IN 1..tarifmodelle(1)'length LOOP
      kosten = cdr_cost(VektorGrundKosten,
                        modell, cdr, tarifmodelle(j));
      IF kosten < kostenTemp THEN
        best_mix = tarifmodelle(j);
        kostenTemp = kosten;
      END IF;
    END LOOP;
  END LOOP;

  RETURN best_mix;
END best_modell;
```

Die Funktion *best_ modell* verwendet die beiden vorher definierten Funktion um den besten Tarifmodellmix zu erstellen. Dieser Algorithmus berücksichtigt dabei alle Teilmengen aus der Menge der Tarifmodelle. Die hier verwendete Funktion getSubSets bestimmt alle i-elementigen Teilmengen einer n-elementigen Menge (dabei werden die Elemente der Menge von 1 bis n durchnummeriert), das Ergebnis wird in einem Array der Dimension $\left[\begin{pmatrix} n \\ i \end{pmatrix}, i\right]$ gespeichert.

```
PACKAGE subset
BEGIN
/**
/*Auf die Menge aller k-elementigen Teilmengen
/*einer n-elementigen Menge wird eine natürliche
/*Ordnung definiert. Anhand dieser Ordnung, kann
/*ein Nachfolger ermittelt werden.
*/
  FUNCTION nachfolger(INT_ARRAY(i): vorgaenger, INT k)
                      RETURN INT_ARRAY(i) IS
  BEGIN
    INT_ARRAY(i) vektor_temp;
    vektor_temp = vorgaenger;
    IF vorgaenger(0) < n THEN
      BEGIN
        vektor_temp(0) = vorgaenger(0) + 1;
        RETURN vektor_temp;
      END
      FOR i IN 1..k-1 LOOP
        IF vorgaenger(i) + 2 <= vorgaenger(i-1) THEN
          vektor_temp(i) = vorgaenger(i) + 1;
          FOR j IN REVERSE 0..i LOOP
            vektor_temp(j-1) = vektor_temp(j) + 1;
          END LOOP;
        END IF;
      END LOOP;
      RETURN vektor_temp;
    END;
  END IF;
    return null;
  END nachfolger;

  FUNCTION binom(INT: n, k) RETURN INT IS
  BEGIN
    IF k = 1 THEN
      RETURN n;
    END IF;
    IF (k = 0) OR (k = n) THEN
      RETURN 1;
    END IF;
```

```
  RETURN (binom(n-1,k-1) + binom(n-1,k));
END binom;

FUNCTION getSubSets(INT n, k) RETURN INT_ARRAY(i,j) IS
BEGIN
  INT_ARRAY(binom(n,k),k): result;
  FOR i IN REVERSE 0..k-1 LOOP
    result(0,i) = i + 1;
  END LOOP;
  FOR i in 1..binom(n,k) - 1 LOOP
    result(i) = nachfolger(result(i-1), n, k);
  END LOOP;
  return result;
END getSubSets;

END subset;
```