➲ AUFGABE 4:VERSTEHST DU BAHNHOF?

JA! Gebäude an dem schienengebundene Fortbewegungsmittel, für Personen und Warentransport, halt machen um ihre Beladung zu wechseln.

Lösungsidee und Programmdokumentation:

Zuerst einmal möchte ich zeigen, wie meine Koppelungsstrategie auf dem Bahnhof funktioniert. Die zwei Loks bleiben immer stehen und werden fest nach Stadt1 oder Stadt2 geschickt. Um die richtige Waggonanzahl zu erhalten, müssen vom Zug1 Waggons zum Zug2 geschoben werden. Man nimmt sich immer eine Gruppe, die zum anderen Zug muss heraus und schiebt sie hinter diesen. Damit dabei keine unnützen Kopplungen vorgenommen werden, lasse ich die Waggongruppen so lange abgekoppelt, bis die richtigen Mengen hinter den Loks vorhanden sind.
Um die günstigste Umkoppelungsvorgehensweise für die Züge herauszufinden, vergleiche ich in meinem Programm alle „sinnvollen“ Umkopplungsvarianten. Dabei gehe ich so vor:
Im String `Zug1`, `Zug2` sind die Waggontypen gespeichert. Zu diesen Strings gibt es zwei boolean Arrays `ausw1` und `ausw2`, welche Waggons die von einem zum anderen Zug verschoben werden sollen markieren. Die Suche besteht aus mehreren rekursiven Funktionen, die einen Möglichkeitenbaum beschreiben. Zu Beginn der Suche kann zuerst unterschieden werden, ob Lok1 nach Stadt1 oder Lok1 nach Stadt2 fahren soll. Dann wird daraus errechnet, wie viel Wagons von den einzelnen Typen zwischen Zug1 und Zug2 wechseln müssen. Sind diese Zahlen berechnet, kommt die Procedure `verteile(mom)` dran. Sie aktiviert die boolean Felder so, dass für einen einzelnen Waggontyp die Anzahl im Zug1 und Zug2 für die Abfahrt zu den einzelnen Städten stimmt. Sind zum Beispiel im Zug1 5 Waggons vom Typ A und die Stadt, in die der Zug fährt, benötigt aber nur 3, so werden 2 der zugehörigen Booleanwerte aktiviert. Es wäre aber nicht gerade sinnvoll alle Aktivierungskombinationen durchzuspielen. Am günstigsten sind die Aktivierungen, bei denen die abzutrennenden Waggons sich an den Randbereichen befinden, da so größere Abkopplungsbereiche entstehen können. In unserem Beispiel wird dadurch die Variante AAAAA und AAAAA getestet. Wenn diese Aktivierung durchgeführt ist, ruft sich die Procedure `verteile` selbst wieder mit einem um eins höheren Parameter auf, damit die Varianten für B,C,… durchgespielt werden. Sind alle Wagontypen durch, so haben die entstandenen Züge schon alle die richtige Waggonanzahl für die einzelnen Städte. Es muss sich aber noch nicht die günstigste Variante darunter befinden.
Dazu ein Beispiel: Zug1 hat die Beladung AACBB, Zug2 DDC. Durch die oben beschriebene Routine kommen nur Verteilungen vor, in denen immer eins der A, B und Ds verschoben werden, wie zum Beispiel AACBB und DDC. Günstiger wäre AACBB und DDC. Deshalb kommt noch eine Procedure `erweitern` hinzu, welche zuerst die Procedure `testen` aufruft und dann sich einen noch nicht markierten Bereich zwischen zwei markierten aussucht, diesen markiert und dafür mit den Funktionen `auswahlzug1` bzw. `auswahlzug2` in dem anderen Zug zur Kompensation andere Wagons auswählt. Die Funktionen `auswahlzug1` bzw. `2` gehen dabei so vor, dass sie immer nur einen Wagon auswählen und sich dann wieder selbst ausführen um einen nächsten Wagon auszuwählen. Dadurch werden alle Kombinationsmöglichkeiten durchlaufen. Wurden ausreichend Wagons zur Kompensation markiert, wird die Procedure erweitern wieder aufgerufen.
Die Procedure testen ermittelt wie viele Abkopplungsvorgänge aus den Markierungen resultieren. Hat die momentane Markierungsvariante eine geringere Abkopplungszahl als die in `minkop` gespeicherte, werden die Arrays `ausw1` und `ausw2` in die Arrays `save1` und `save2` übertragen. Die Anzahl der Abkopplungsvorgänge ist logischerweise immer die Hälfte der gesamten Kopplungsvorgänge.
Da durch die Rekursionen Programmteile aufgerufen werden müssen, welche noch nicht definiert sind, musste ich die Suche nach der günstigsten Abkopplungsvariante in ein UNIT auslagern.

Quelltext des Units (UNIT4.PAS):

```
UNIT unit4;

INTERFACE
VAR ausw1, ausw2, save1, save2: array[0..60] of boolean;     {momentane+gesp. Markierungen}
    zug1, zug2:string;                                                {Waggonabfolge}
    typ : array[1..10] of char;                                {Chars der verschiedenen Typen}
    neu1, neu2, alt1, alt2, versch: array[1..10] of shortint; {Anzahl der einzelnen Typen}
    typanz: byte;                                                    {Anzahl der Typen}
    minkop:byte;                                                    {minimale Zuganzahl}
    wzug, wzugs:boolean;                                            {Zug1->Stadt1=false}
PROCEDURE erweitern;
PROCEDURE suche;
```

```
IMPLEMENTATION

PROCEDURE testen;
VAR i,j, kopplungen: byte;
BEGIN
kopplungen := 0;                                {Test an wievielen Stellen abgekoppelt wird}
FOR i := 0 to length(zug1)-1 do if ausw1[i]<>ausw1[i+1] then inc(kopplungen);
FOR i := 0 to length(zug2)-1 do if ausw2[i]<>ausw2[i+1] then inc(kopplungen);

if kopplungen < minkop then BEGIN          {Wenn der momentane Koppelvorgang günstiger ist}
 minkop := kopplungen;                                          {wird dieser gespeichert}
 save1 := ausw1;
 save2 := ausw2;
 wzugs := wzug;
END;
END;

PROCEDURE auswahlzug1(a, e: byte);        {Wählt im Zug1 Waggons zum abkoppeln aus die den}
VAR i: byte;                             {Inhalt der Wagons a bis a+e vom Zug2 entsprechen}
BEGIN
FOR i := 1 to length(zug1) do if not(ausw1[i]) and (zug2[a]=zug1[i]) then BEGIN
ausw1[i] := true;
if e>1 then auswahlzug1(a+1, e-1)                              {nach nächsem Waggontyp suchen}
 else erweitern;            {Wenn ausreichend Wagons markiert sind wird wieder verglichen}
ausw1[i] := false;
END;
END;

PROCEDURE auswahlzug2(a, e: byte);        {Wählt im Zug2 Waggons zum abkoppeln aus die den}
VAR i: byte;                             {Inhalt der Wagons a bis a+e vom Zug1 entsprechen}
BEGIN
FOR i := 1 to length(zug2) do if not(ausw2[i]) and (zug1[a]=zug2[i]) then BEGIN
ausw2[i] := true;
if e>1 then auswahlzug2(a+1, e-1)                              {nach nächsem Waggontyp suchen}
 else erweitern;            {Wenn ausreichend Wagons markiert sind wird wieder verglichen}
ausw2[i] := false;
END;
END;

PROCEDURE erweitern;      {Versuch bessere Lösungen zu finden, indem zwischen den Berechen}
VAR a, e, i: byte ;     {welche abgekoppelt werden auch anderen Waggons abgekoppelt werden}
BEGIN                                          {und dafür vom anderen Zug zurückkommen}
testen;

a := 255;
FOR e := 1 to length(zug1) do BEGIN            {Suchen nach einem nicht markiertem Bereich}
 if not(ausw1[e]) then BEGIN if a>e then a:= e END
  ELSE BEGIN
   for i := a to e-1 do ausw1[i] := true;
   auswahlzug2(a, e-a);                          {Kompensation: Waggons im Zug2 aktivieren}
   for i := a to e-1 do ausw1[i] := false;
   a := 255;
 END;
END;

a := 255;
FOR e := 1 to length(zug2) do BEGIN            {Suchen nach eimen nicht markiertem Bereich}
 if not(ausw2[e]) then BEGIN if a>e then a:= e END
  ELSE BEGIN
   for i := a to e-1 do ausw2[i] := true;
   auswahlzug1(a, e-a);                          {Kompensation: Waggons im Zug1 aktivieren}
   for i := a to e-1 do ausw2[i] := false;
   a := 255;
 END;
END;

END;


PROCEDURE verteile(mom: byte);                 {Waggons markieren damit die Anzahl zu den}
VAR i : shortint;                                          {verschieden Städten stimmt}
BEGIN
if mom <= typanz then BEGIN
if versch[mom] = 0 then verteile(mom+1);
if versch[mom] > 0 then BEGIN                          {Wenn Zug1 zuviel vom Typ mom hat}
  for i := pos(typ[mom],zug1) to pos(typ[mom],zug1)+versch[mom]-1 do ausw1[i] := true;
  verteile(mom+1);                                                  {Nächster Waggontyp}
```

```
  for i := pos(typ[mom],zug1) to pos(typ[mom],zug1)+versch[mom]-1 do ausw1[i] := false;

  if alt1[mom] <> versch[mom] then begin                    {rechte Waggons markieren}
  for i := pos(typ[mom],zug1)+alt1[mom]-versch[mom] to pos(typ[mom],zug1)+alt1[mom]-1 do
      ausw1[i] := true;
  verteile(mom+1);                                                {Nächster Waggontyp}
  for i := pos(typ[mom],zug1)+alt1[mom]-versch[mom] to pos(typ[mom],zug1)+alt1[mom]-1 do
      ausw1[i] := false;
  end;
END;

if versch[mom] < 0 then BEGIN                          {Wenn Zug2 zuviel vom Typ mom hat}
  for i := pos(typ[mom],zug2) to pos(typ[mom],zug2)-versch[mom]-1 do ausw2[i] := true;
  verteile(mom+1);                                                {Nächster Waggontyp}
  for i := pos(typ[mom],zug2) to pos(typ[mom],zug2)-versch[mom]-1 do ausw2[i] := false;

  if alt2[mom] <> -versch[mom] then begin                   {rechte Waggons markieren}
  for i := pos(typ[mom],zug2)+alt2[mom]+versch[mom] to pos(typ[mom],zug2)+alt2[mom]-1 do
      ausw2[i] := true;
  verteile(mom+1);                                                {Nächster Waggontyp}
  for i := pos(typ[mom],zug2)+alt2[mom]+versch[mom] to pos(typ[mom],zug2)+alt2[mom]-1 do
      ausw2[i] := false;
  end;
 END
END
else BEGIN
erweitern;
END;

END;


PROCEDURE suche;                        {Suche nach der günstigsten Kopelvariante starten}
VAR gefunden : boolean;
    i:byte;
BEGIN
 {Lok1 fährt zur Stadt1}
 for i := 1 to typanz do versch[i] := alt1[i] - neu1[i]; {Berechnung wieviel Waggons vom}
 for i := 0 to 60 do BEGIN ausw1[i] := false; ausw2[i] := false; end; {Zug 1 zu 2 muessen}
 wzug := false;
 verteile(1);                                                          {Rekursion starten}

 {Lok1 fährt zur Stadt2}
 for i := 1 to typanz do versch[i] := alt1[i] - neu2[i];
 for i := 0 to 60 do BEGIN ausw1[i] := false; ausw2[i] := false; end;
 wzug := true;
 verteile(1);                                                          {Rekursion starten}
END;


BEGIN
minkop := 255;                                          {maximale Koppelanzahl setzen}
END.
```

Hauptteil der Programmes (AUFGABE4.PAS):

Da im Unit die ganze Suchroutine untergebracht ist, brauche ich im Hauptteil nur noch die Ein- und Ausgabe unterbringen. Bei der Eingabe werden zuerst die zwei Stings `Zug1` und `Zug2` eingelesen, dann darauf untersucht welche Buchstaben vorkommen und danach die Anzahl der Wagons für jeden Buchstaben. Nur Großbuchstaben sind erlaubt. Darauf wird für jeden Buchstaben die Anzahl für die Stadt1 eingegeben. Die Zahlen für Stadt2 berechnet er selbst und schreibst sie auf den Bildschirm. Danach wird die Suche durchgeführt. Die Ausgabe Procedure gibt zuerst die Zahl der gesamten Koppelvorgänge aus. Dann die beste Variante im Programmnahem Markierungsstil. Wenn die Wagons markiert sind (grün) müssen sie durch umkoppeln zum anderen Zug. An dieser Variante kann man besonders schön die Funktionsfähigkeit sehen. Dann kommt die geforderte Variante mit den Koppelanweisungen. Zuerst die Abkopplungspunke vom Zug1 und die zugehörigen Verschiebungsinformationen, dann die vom Zug2. (Wenn ich der Bahnhofsvorsteher wäre, würde mir die Variante mit den grünen Markierungen einfacher fallen.) Danach gibt das Programm noch die entstandene Wagonabfolge der Züge aus und gibt dem Bahnhofsvorsteher die Informationen, ob er die Lok1 zur Stadt1 oder Stadt2 schicken soll.

Quelltext (AUFGABE4.PAS):

```
PROGRAM Bahnhof;
USES crt, unit4;

VAR i, k: integer;


PROCEDURE EINGABE;
BEGIN

WRITE('Zug1: '); READLN(zug1);                              {Eingabe der 2 Wagonabfolgen}
WRITE('Zug2: '); READLN(zug2);
typanz := 0;
for i := 65 to 90 do                        {Alle Grosbuchstaben durchgehn und für jeden Typ}
 if (pos(chr(i),zug1) <> 0) or  (pos(chr(i),zug2) <> 0) then BEGIN {einen Counter anlegen}
 inc(typanz);
 typ[typanz] := chr(i);
 alt1[typanz] := 0;  alt2[typanz] := 0;
 END;
                                                  {Anzahl der einzelnen Wagons feststellen}
FOR i := 0 to length(zug1) do for k := 1 to typanz do if typ[k]=zug1[i] then inc(alt1[k]);
FOR i := 0 to length(zug2) do for k := 1 to typanz do if typ[k]=zug2[i] then inc(alt2[k]);

WRITELN;
WRITELN('Tagesbedarf fuer Stadt1:');
FOR i := 1 to typanz DO BEGIN                    {Eingabe wieviele Wagons von welchem Typ}
 WRITE(typ[i],':');                                                {der Zug 1 erhalten soll}
 READ(neu1[i]);
 GOTOXY(i*6, WHEREY-1);
 END;
WRITELN;

WRITELN('Tagesbedarf fuer Stadt2:');          {Berechnung wieviele für Zug 2 ueberbleiben}
FOR i := 1 to typanz DO BEGIN
 WRITE(typ[i],':');
 neu2[i] := alt1[i] + alt2[i] - neu1[i];
 WRITELN(neu2[i]);
  GOTOXY(i*6, WHEREY-1);
 END;
WRITELN;
END;


PROCEDURE ausgabe;
BEGIN
 WRITELN;
 WRITELN('Es sind genau ',minkop*2,' Koppelvorgaenge notwendig!');
 WRITELN('Die gruenen Wagons werden abgetrennt und hinter den anderen Zug gestellt.');

 FOR i := 1 to length(zug1) do BEGIN             {Ausgabe Zug1, abzutrennende Wagons grün}
  if save1[i] then textcolor(2) else textcolor(7);
  write(zug1[i]);
 END;
 WRITELN;

 FOR i := 1 to length(zug2) do BEGIN                                {Das selbe für den Zug 2}
  if save2[i] then textcolor(2) else textcolor(7);
  write(zug2[i]);
 END;

 textcolor(7);
 WRITELN;
 WRITELN;
 Write('Zug1 ');                                    {Eine Koppelanweisung als Text ausgeben}
 FOR i := 0 to length(zug1)-1 do if save1[i]<>save1[i+1] then BEGIN
  WRITE('zwischen ', i, ' und ', i+1, ' abkoppeln und hinter ');
  if save1[i] then write('Zug1 stellen, dann ')
     else write('Zug2 stellen, dann ');
  END;
 WRITELN('kommt Zug2 dran.');
 Write('Zug2');
 FOR i := 0 to length(zug2)-1 do if save2[i]<>save2[i+1] then BEGIN
  WRITE(' zwischen ', i, ' und ', i+1, ' abkoppeln und hinter ');
  if save1[i] then write('Zug2 stellen, dann ')
     else write('Zug1 stellen, dann ');
  END;
 Writeln('von hinten auf die Waggons hinter Zug1 und Zug2 mit einer Rangierlok ',
         'drauffahren und die Waggons zusammenkoppeln.');
```

```
 WRITELN;
 WRITELN('Dadurch entstehen diese Zuege:');        {Ausgabe der dadurch entstandenen Züge}
 FOR i := 1 to length(zug1) do if not(save1[i]) then write(zug1[i])
                               else if not(save1[i-1]) then write(' ');
 WRITE(' ');
 FOR i := 1 to length(zug2) do if save2[i] then write(zug2[i])
                               else if save2[i-1] then write(' ');
 WRITELN;

 FOR i := 1 to length(zug2) do if not(save2[i]) then write(zug2[i])
                               else if not(save2[i-1]) then write(' ');
 WRITE(' ');
 FOR i := 1 to length(zug1) do if save1[i] then write(zug1[i])
                               else if save1[i-1] then write(' ');
 WRITELN;
 WRITELN;
 if wzugs then
 WRITELN('Lock1 muss in Stadt2 fahren!')
 else WRITELN('Lock1 muss in Stadt 1 fahren!');

END;


BEGIN
 clrscr;
 EINGABE;
 SUCHE;
 AUSGABE;

 REPEAT UNTIL keypressed;
END.
```

3 Test Beispiele:

1. Vorgegebenes Beispiel:

```
Zug1: AAAAAAAABBBBBBCCCCCCDDD
Zug2: DDDDDEEEEEEECCCCCCCCCAAAA

Tagesbedarf fuer Stadt1:
A:5  B:0   C:7   D:8   E:3
Tagesbedarf fuer Stadt2:
A:7  B:6   C:8   D:0   E:4

Es sind genau 10 Koppelvorgaenge notwendig!
Die gruenen Wagons werden abgetrennt und hinter den anderen Zug gestellt.
AAAAAAAABBBBBBCCCCCCDDD
DDDDDEEEEEEECCCCCCCCCAAAA

Zug1 zwischen 0 und 1 abkoppeln und hinter Zug2 stellen, dann zwischen 1 und 2 a
bkoppeln und hinter Zug1 stellen, dann zwischen 20 und 21 abkoppeln und hinter Z
ug2 stellen, dann kommt Zug2 dran.
Zug2 zwischen 8 und 9 abkoppeln und hinter Zug1 stellen, dann  zwischen 14 und 1
5 abkoppeln und hinter Zug1 stellen, dann von hinten auf die Waggons hinter Zug1
 und Zug2 mit einer Rangierlok drauffahren und die Waggons zusammenkoppeln.

Dadurch entstehen diese Zuege:
 AAAAAAABBBBBBCCCCCC  EEEECC
DDDDDEEE CCCCCCCAAAA A DDD

Lock1 muss in Stadt2 fahren!
```

2. Beispiel:

```
Zug1: AADBB
Zug2: CCCD

Tagesbedarf fuer Stadt1:
A:1  B:1   C:1   D:1
Tagesbedarf fuer Stadt2:
A:1  B:1   C:2   D:1

Es sind genau 6 Koppelvorgaenge notwendig!
Die gruenen Wagons werden abgetrennt und hinter den anderen Zug gestellt.
AADBB
CCCD

Zug1 zwischen 1 und 2 abkoppeln und hinter Zug2 stellen, dann zwischen 4 und 5 a
bkoppeln und hinter Zug1 stellen, dann kommt Zug2 dran.
Zug2 zwischen 2 und 3 abkoppeln und hinter Zug2 stellen, dann  zwischen 4 und 5
abkoppeln und hinter Zug2 stellen, dann von hinten auf die Waggons hinter Zug1 u
nd Zug2 mit einer Rangierlok drauffahren und die Waggons zusammenkoppeln.

Dadurch entstehen diese Zuege:
A B CD
CC  ADB

Lock1 muss in Stadt 1 fahren!
```

3. Beispiel

```
Zug1: AABBBBBBCCCCDDDDDEEEEEFFFF
Zug2: FFFFFAAAABBBCCCCDDDEEE

Tagesbedarf fuer Stadt1:
A:1  B:3   C:7   D:6   E:4   F:2
Tagesbedarf fuer Stadt2:
A:5  B:6   C:1   D:2   E:4   F:7
Es sind genau 12 Koppelvorgaenge notwendig!
Die gruenen Wagons werden abgetrennt und hinter den anderen Zug gestellt.
AABBBBBBCCCCDDDDDEEEEEFFFF
FFFFFAAAABBBCCCCDDDEEE

Zug1 zwischen 0 und 1 abkoppeln und hinter Zug2 stellen, dann zwischen 9 und 10
abkoppeln und hinter Zug1 stellen, dann zwischen 21 und 22 abkoppeln und hinter
Zug2 stellen, dann zwischen 24 und 25 abkoppeln und hinter Zug1 stellen, dann ko
mmt Zug2 dran.
Zug2 zwischen 8 und 9 abkoppeln und hinter Zug2 stellen, dann  zwischen 17 und 1
8 abkoppeln und hinter Zug1 stellen, dann von hinten auf die Waggons hinter Zug1
 und Zug2 mit einer Rangierlok drauffahren und die Waggons zusammenkoppeln.

Dadurch entstehen diese Zuege:
 CCCDDDDDEEEE FF ABBBCCCCD
FFFFFAAA DDEEE AABBBBBBC EFF

Lock1 muss in Stadt 1 fahren!
```