3.1 Kürzeste Wege

Während beim Traveling Salesman Problem die kürzeste Rundreise über alle Städte gesucht ist, soll beim *Kürzesten-Wege-Problem* die kürzeste Verbindung zwischen zwei Städten berechnet werden. Ist die Entfernungsmatrix metrisch, so ist dies wegen der Gültigkeit der Dreiecksungleichung immer die direkte Verbindung, bei nichtmetrischer Aufgabenstellung kann es jedoch auch günstiger sein, Zwischenstationen zu benutzen. Zur Lösung kann der Algorithmus von Dijkstra verwendet werden, der in der folgenden Form für einen Graphen mit der Knotenmenge V und der Kostenmatrix ENTFTAB die Längen der kürzesten Wege zwischen einem wählbaren Startpunkt ANFANG und allen anderen Knoten im Feld LAENGE liefert:

```
1: LAENGE[ANFANG]:=0; T:=V;
2: für I ∈ V\{ANFANG} LAENGE[I]:=∞;
3: solange T ≠ ∅
4:  finde J ∈ T mit LAENGE[J] minimal;
5:  T:=T\{J};
6:  für K ∈ T LAENGE[K]:=min(LAENGE[K],LAENGE[J]+ENTFTAB[J,K]);
7: ende{solange}
```

In Zeile 6 wird untersucht, ob es möglich ist, den bisherigen Weg von ANFANG nach K durch Einfügen des Weges von ANFANG nach J zu verkürzen. Ist man neben den Längen auch an den Wegen selbst interessiert, so muß zusätzlich zu LAENGE noch ein WEGE-Array geführt werden. Wird dann tatsächlich eine Verkürzungsmöglichkeit gefunden, so ist der neue Teilweg an der entsprechenden Stelle einzubauen.

Im Programm 3.1 werden im Gegensatz zum angegeben Algorithmus die Elemente von LAENGE nicht mit ∞, sondern mit den Entfernungen für die Direktverbindungen, initialisiert und die Wege entsprechend vorgegeben. Zu Anfang wird die Startstadt mit der Maus ausgewählt, nach der Berechnung können mit der linken Maustaste die Ergebnisse für die einzelnen Zielorte abgefragt und mit der rechten die Ausgabe beendet werden.

3.2 Minimal Spanning Trees

Läßt man bei einer Tour den Rückweg von der letzten Stadt zum Ausgangspunkt weg, so erhält man einen erzeugenden Baum. Die Summe der Kosten des minimalen erzeugenden Baumes T stellt folglich eine untere Schranke für die Länge aller Touren eines gegebenen TSP dar. Berechnet werden kann er für einen Graphen mit der Punktemenge V={1,...,N} und der Kostenmatrix ENTFTAB mit dem Algorithmus von Prim:

```
 1: W(1):=0; S:=V; T:=∅;
 2: für I:=2 to N W(I):=∞;
 3: solange S ≠ ∅
 4:  bestimme I ∈ S mit maximalem W(I);
 5:  S:=S\{I};
 6:  falls E(I) definiert ist dann T:=T∪{E(I)};
 7:  für E ∈ {Kanten mit Startpunkt in V\S und Endpunkt in S}
 8:   V:=Endpunkt von E in S;
 9:   falls W(V)>ENTFTAB[Startpunkt von E,Endpunkt von E] dann
10:    W(V):=ENTFTAB[Startpunkt von E,Endpunkt von E];
11:    E(V):=E;
12:   ende{falls}
13:  ende{für}
14: ende{solange}
```

Im Programm wird der Baum nicht in einer Menge von Kanten, sondern in einer Variable vom Typ WEG (d. h. einem String) in Form der Stadtnummern gespeichert, wobei die Kanten durch Nullen voneinander getrennt sind, es sei denn, der Endpunkt einer Kante und der Anfangspunkt

der nächsten sind gleich. In diesem Fall sorgt Programmzeile 29 dafür, daß die Stadt nur einmal und ohne Null eingefügt wird.

3.3 1-Bäume

Bei der Berechnung einer unteren Schranke für ein Traveling Salesman Problem durch einen Minimal Spanning Tree wird durch das Weglassen einer Kante etwas verschenkt. Dies ist nicht der Fall, wenn man statt dessen *1-Bäume* betrachtet. Da eine Tour ein spezieller 1-Baum ist, kann die Länge des minimalen 1-Baumes ebenfalls als untere Schranke dienen. Zur Berechnung wird erst ein Minimal Spanning Tree auf dem Graph ohne die herausgenommene Stadt ermittelt und diese dann mit den zwei kürzesten Kanten hinzugefügt.

3.4 Permutationen

Die Menge aller Touren eines TSP ist durch die Permutationen der n Städte und Hinzufügen des Rückweges zum Start gegeben. Da für ein festes n ihre Anzahl endlich ist, läßt sich die optimale Tour prinzipiell durch Überprüfung aller möglichen Lösungen ermitteln. Die Bestimmung der Permutationen kann rekursiv erfolgen, wobei sich mit jeder Rekursionsebene der Weg um eine Stadt verlängert.

Es wäre naheliegend, beim Aufruf der Prozedur PERM den bisherigen Weg mit zu übergeben, allerdings hätte dies einen sehr hohen Speicherplatzbedarf auf dem Stack zufolge. Betrachtet man jedoch die Reihenfolge, in der der Rekursionsbaum aufgebaut wird, so erkennt man, daß eine Stadt auf einer höheren Rekursionsebene erst dann geändert wird, wenn alle von dort ausgehenden Verzweigungen der nächsten Ebenen bereits untersucht wurden. Es genügt daher, eine einzige Variable zu verwalten und sie lediglich beim Zurückgehen in eine frühere Ebene um eine Stadt zu verkürzen.

Grundsätzlich ist es mit diesem Programm möglich, jedes Traveling Salesman Problem zu lösen, wenn man nur ausreichend lange wartet. In der Zwischenzeit kann man jedoch einmal berechnen, wie lange diese Wartezeit ist: Die Anzahl der Permutationen für n Städte beträgt $n!$, d. h. die Zahl der zu untersuchenden Touren und damit die Rechenzeit wächst bei der Erhöhung von n auf $(n+1)$ Städte auf das $(n+1)$-fache. Auf einem Testrechner läßt sich ein bestimmtes 10-Städte-Probleme in 133,19 s lösen. Für 15 Städte kommt man dann auf ca. 556 Tage, bei 50 Städten wären es mehr als $3 \cdot 10^{52}$ Jahre.