Programmieren C: Backtracking: Travelling Salesman

Praktisch:

Gegeben sind *Städte auf einer Landkarte* und die *Länge der Straßen dazwischen*. Gesucht ist die *kürzeste Rundreise*, die *alle Städte* besucht und zur Ausgangsstadt zurückkehrt, wobei jede Stadt *nur einmal besucht* und jede Straße *nur einmal befahren* werden darf.

Theoretisch:

Travelling Salesman ist ein *graphentheoretisches Problem*, das man mittels Backtracking lösen kann (das Problem ist *NP-vollständig*, d.h. ein wesentlich "effizienteres" Verfahren zu seiner Lösung ist (noch) nicht bekannt). Es geht bei diesem Problem darum, in einem *ungerichteten, kantengewichteten Graphen* die *kürzeste Rundreise* zu finden, d.h. diejenige Teilmenge der Kanten, die einen Zyklus darstellt, der alle Knoten des Graphen genau einmal durchläuft und in Summe minimales Gewicht hat.

Du kannst dir ein ***Programm herunterladen***, das bereits das *Hauptprogramm* und *alle globalen Variablen* enthält. Dieses Hauptprogramm erzeugt zufällig Städte und einige Straßen dazwischen, ruft eine *Funktion zur Ermittlung der besten Rundreise* auf, und gibt dann die gefundene Lösung aus.

Mach Dich zuerst mit dem Programm und vor allem seinen globalen Daten vertraut und schreibe dann diese ***Lösungs-Such-Funktion*** (*rekursiv mittels Backtracking*):
Das Backtracking wird *pro Aufrufsebene eine Reise-Etappe bzw. eine Stadt* behandeln:
Ausgehend von der gerade aktuellen Stadt sind alle anderen Städte als mögliche nächste Etappe der Reise zu prüfen.

Im Detail:

- Die Funktion hat *drei Parameter*:
  - Die *wievielte Etappe* führt uns zur aktuellen Stadt, d.h. *beim wievielten Zwischenstopp* besuchen wir die aktuelle Stadt? (beim äußersten Aufruf 0)
  - In der *wievielten Stadt* befinden wir uns gerade? Diese Stadt-Nummer dient *als Index* in die globalen Arrays. (beim äußersten Aufruf 0, unsere Reise *beginnt und endet* immer in *Stadt 0*)
  - *Wie lange ist die Strecke*, die wir bis hierher insgesamt zurückgelegt haben? (ebenfalls beim äußersten Aufruf 0)
- Als erstes sollte die Funktion gleich einmal *in der aktuellen Lösung speichern*, dass wir die *aktuelle Stadt als x-ten Zwischenstopp* besuchen.

  Überlege: Was muss die Funktion *als allerletztes tun*, *bevor* sie wieder *zurückkehrt*?
- Wenn die aktuelle Etappe die *vorletzte Etappe* ist, müssen wir *nicht weiter suchen*, denn die *letzte Etappe* muss *immer zurück zur Stadt 0* führen.

  Zähle daher die *Entfernung zur Stadt 0* zur bisherigen Wegstrecke dazu, prüfe, ob das Ergebnis ein *neues Optimum* ist, und wenn ja, *speichere* die gesamte aktuelle Route als *neue beste Route* und ihre Länge als neue beste Länge.

- Wenn die aktuelle Etappe *nicht die vorletzte* ist, dann müssen wir mit einer Schleife *alle Städte für die nächste Etappe* durchprobieren.
  In Frage kommen dabei nur jene Städte, die *bisher noch nicht besucht* wurden.

  Für jede dieser Städte wird die *neue Wegstrecke berechnet*,
  und nur dann, wenn diese Strecke noch *kürzer als die bisher beste Lösung ist*,
  wird diese Route durch einen *rekursiven Aufruf* mit der neuen Stadt weiterverfolgt.
- Eigentlich müsste man sowohl für die *nächste* als auch für die *letzte* Etappe *prüfen*, ob überhaupt eine *direkte Straße* von der aktuellen Stadt zur nächsten Stadt beziehungsweise zurück zur Stadt 0 führt.

  Diese Prüfung kann man aber *weglassen*: Für nicht vorhandene Straßen ist *als Entfernung "unendlich"* gespeichert. Die bisherige Wegstrecke plus "unendlich" ergibt "unendlich" als neue Gesamt-Wegstrecke, und eine *unendlich lange Lösung* wird in beiden Fällen vom unmittelbar darauffolgenden **`if`** *sofort verworfen*.

Andere Hinweise:

- Da unsere "Landkarte" ja jedesmal *zufällig erzeugt* wird und ***nicht** alle möglichen Verbindungen von jeder Stadt zu jeder anderen Stadt* enthält, kann es passieren, dass *gar keine Lösung* existiert (z.B. wenn die Städte in zwei Bereiche zerfallen, die nur über eine einzige Straße, eine einzige Stadt oder *gar nicht verbunden* sind).
- Durch Ändern der **`#define`**-Konstante **`MIN_VERB`** lässt sich kontrollieren, *wie viele Straßen* zwischen den Städten erzeugt werden.
- Für Linux- und Mac-Benutzer: Das Programm verwendet Funktionen aus **`math.h`** . Beim Linken ist daher die Option **`-lm`** nötig.