Programmieren C++: Mehrdim. Arrays, Arrays von Pointern

Klaus Kusche

Entfernungen optimieren (Floyd-Warshall-Verfahren)

Gesucht ist eine Klasse **`Wege`**, die die optimalen Fahr-Distanzen zwischen Städten liefert.
Ich stelle ein Hauptprogramm zum Testen zur Verfügung.

Die Klasse hat folgende *Member-Variablen*:

- Die *Anzahl* der Städte.
- Eine *dynamisch angelegte Matrix der Distanzen* zwischen allen Paaren von Städten (Zeile ist die Ausgangs-Stadt, Spalte ist die Ziel-Stadt, Wert ist die Entfernung zwischen den beiden Städten, Größe der Matrix ist Städtezahl * Städtezahl). Für nicht bekannte Verbindungen wird der Wert **`HUGE_VAL`** (unendlich) gespeichert.

  Da sich zweidimensionale Arrays variabler Größe nicht mit **`new`** anlegen lassen, ist die Member-Variable ein *Pointer auf ein Array von Pointern*, das die Zeilen darstellt. Jeder Pointer in diesem Array zeigt auf ein eindimensionales Array von **`double`**-Werten, das die Werte der Spalten in dieser Zeile enthält.
- Ein **`bool`**-sches Flag, das anzeigt, ob schon die *optimalen* Entfernungen berechnet wurden.
- Einen *Pointer auf ein Array von Pointern auf Stringkonstanten*, das die Städte-Namen (zur schönen Beschriftung der Ausgabe) enthält.

Weiters hat die Klasse folgende *Methoden*:

- Einen *Konstruktor*, der die *Anzahl der Städte* und einen Pointer auf deren *Namens-Array* als Argumente übergeben bekommt.

  Er muss das Zeilen-Array sowie die Spalten-Arrays der Matrix *dynamisch anlegen* und dann alle Entfernungen auf **`HUGE_VAL`** initialisieren.

  Der Pointer auf das Namens-Array kann direkt als Member gespeichert werden, das Namens-Array muss nicht kopiert werden.
- Einen *Destruktor*, der alle dynamisch angelegten Arrays wieder freigibt.
- Eine *Get-Methode* für die *Anzahl* der Städte.
- Eine *Get-Methode* für die *Distanz*, die mit der Nummer der von-Stadt und der Nummer der nach-Stadt aufgerufen wird und die Entfernung zurückliefert (bzw. **`HUGE_VAL`** bei ungültigen Städte-Nummern).
- Eine *Set-Methode* für die *Distanz* zweier Städte: Sie wird mit der Nummer der von-Stadt und der Nummer der nach-Stadt sowie der Entfernung aufgerufen.

  Bei Erfolg liefert sie **`true`** zurück, bei Fehler (ungültige Städte-Nummer oder Distanz kleiner 0) **`false`**.

  Das Ändern der Entfernungen darf nur möglich sein, solange die kürzesten Distanzen *noch nicht berechnet* wurden. Danach dürfen die Werte nicht mehr verändert werden, die Set-Methode kehrt daher in diesem Fall mit **`false`** zurück.

- Eine Methode, die für alle Paare von Städten die *kürzesten Distanzen berechnet* (und das in der Optimierungs-Membervariable vermerkt).

  Dafür wird das Verfahren von Floyd-Warshall verwendet, das aus *drei ineinander geschachtelten* **for**-*Schleifen* besteht (d.h. *alle möglichen Kombinationen von drei Städten* durchgeht):

  - Die äußerste Schleife durchläuft der Reihe nach alle Städte als *Zwischenhalt*.

    (Es ist wichtig, dass die *Schleife für den Zwischenhalt außen* ist, sonst liefert das Verfahren falsche Ergebnisse!)
  - Die mittlere Schleife durchläuft der Reihe nach alle Städte als *von-Stadt*.
  - Die innerste Schleife durchläuft der Reihe nach alle Städte als *nach-Stadt*.
  - In der innersten Schleife wird geprüft, ob der Weg von der von-Stadt zum Zwischenhalt und von dort zur nach-Stadt (d.h. die Summe der Matrix-Elemente von-Stadt/Zwischenhalt plus Zwischenhalt/nach-Stadt) *kürzer* ist als die derzeit in der Matrix gespeicherte direkte Entfernung zwischen von-Stadt und nach-Stadt.

    Wenn ja, wird die Summe der Entfernungen von-Stadt --› Zwischenhalt --› nach-Stadt als neue direkte Entfernung zwischen von-Stadt und nach-Stadt in der Matrix *gespeichert*.

Schließlich ist noch eine Funktion **operator<<** zur Ausgabe aller Distanzen in Matrix-Form (samt Beschriftung mit den Städte-Namen) auf einen **ostream** zu implementieren (wie musst du diese Funktion deklarieren, damit du direkt auf die Namen und Entfernungen zugreifen kannst?).

Hinweise:

- Alle Entfernungen sind **double**.
- **HUGE_VAL** (für unbekannte Entfernungen) kommt aus dem Header **cmath**.

  Bei der Ausgabe wird für **HUGE_VAL** automatisch **inf** ausgegeben, du musst diesen Fall nicht extra behandeln.
- Du sparst dir wieder viele Prüfungen auf kleiner 0, wenn du statt **int** überall **unsigned int** verwendest.
- **Wege(const Wege &w)** und **Wege &operator=(const Wege &w)** solltest du **private** *deklarieren und undefiniert lassen*, damit es nicht möglich ist, versehentlich ein Wege-Objekt zu kopieren oder zuzuweisen.
- Zum Ausgeben der Entfernungen in Tabellen-Form brauchst du den Header **iomanip** und folgende Funktionen:

  cout << fixed << setw(9) << setprecision(1) << … **;**

  Das gibt den **double**-Wert … in *Fixkomma-Darstellung* mit 9 Zeichen Mindestbreite und 1 Nachkomma-Stelle aus.

  Mit **cout << setw(9) <<** … **;** kannst du auch einen String … mit fixer *Mindestbreite* ausgeben, mit **cout << setw(9) << left <<** … **<< right ;** macht er dasselbe *linksbündig* und schaltet dann wieder auf rechtsbündig um.