Inf Progtech Übung: Kruskal's Algorithmus

Klaus Kusche

Implementiere Kruskal's Algorithmus.

Genauer:

- Aufgerufen wird das Programm mit der *Anzahl der Städte*, die es verbinden soll.
- Als erster Schritt sind *in einem rechteckigen "Land"* die gegebene Anzahl von "Städten" *zufällig angeordnet* zu erzeugen.

  Die Städte sollen mittels SDL (Download und Installationsanleitung bei der Übungsangabe) auch *grafisch dargestellt* werden (z.B. mit einem Kreuzerl). Die zufälligen Koordinaten der Städte sind daher Pixel-Koordinaten (also ganze Zahlen zwischen **`0`** und **`SDL_X_SIZE-1`** bzw. **`SDL_Y_SIZE-1`**, abzüglich ein paar Pixel *Sicherheitsabstand* rundherum, damit nicht über den Rand hinaus gezeichnet wird).
- Als zweiter Schritt ist die *Länge aller möglichen Verbindungen* (Luftlinie: *Pythagoras!*) zwischen den Städten zu berechnen (als **`double`**-Zahlen) und zu *sortierten*.
- Dann ist auf diese Verbindungen *Kruskal's Algorithmus* anzuwenden. Die von diesem ausgewählten *Verbindungen* sind *grafisch anzuzeigen* (Linie).
- *Ausgegeben* soll die *Gesamtlänge aller ausgewählten Verbindungen* werden.

Hinweise:

- Definiere dir je eine *Struktur* für die *Städte* (Pixel-Koordinaten, "Vater-Pointer" im Teilbaum für die Union-Find-Datenstruktur des Kruskal-Algorithmus) und eine für die *Verbindungen* (Pointer auf die beiden Endpunkte bzw. Städte, Länge).
- Lege die benötigten Arrays *dynamisch* in der richtigen Größe an. **Achtung:** Die Verbindungen sind *ungerichtet*! Achte darauf, für jedes Paar von Städten nur *eine* Verbindung zu erzeugen und nicht zwei.
- Du brauchst zur Auswahl der nächsten Kante *kein* komplexes Verfahren implementieren: Du kannst einmal vorab alle Kanten mit der *vordefinierten* **`qsort`**-*Funktion* nach Länge sortieren und dann das sortierte Array einfach der Reihe nach durchgehen. Implementiere die für **`qsort`** nötige Vergleichs-Funktion! (Üblich wäre, die Kanten *als Heap* zu verwalten).
- Für die Prüfung, ob eine Kante innerhalb eines Teilbaumes liegt oder zwei verschiedene Teilbäume verbindet, solltest du zumindest in Ansätzen die übliche effiziente Methode implementieren: Eine *Union-Find-Struktur*, siehe z.B. Wikipedia (bei Unklarheiten: mich fragen!). Die Optimierungen (Größenheuristik, Pfad-Kompression) musst du nicht implementieren.

Anmerkung:

Für einen (annähernd) vollständigen Graphen berechnet der *Algorithmus von Prim* den minimalen aufspannenden Baum normalerweise viel effizienter als der von Kruskal. Du könntest daher auch versuchen, den Algorithmus von Prim zu implementieren. Er ist allerdings bei effizienter Implementierung betreffend Datenstrukturen und Code deutlich komplizierter als der von Kruskal (braucht einen Heap).