Übung 10

Abgabe: 03. Juli 2023, 18:00 Uhr

Aufgabe 10.1: Topologische Sortierung [10 Punkte, Link zur Abgabe] Schreiben Sie ein Programm, das einen gerichteten Graphen $G = (V, E)$ einliest und eine topologische Sortierung dieses Graphen (siehe Aldat Skript Listing 8.5 in $O(|V| + |E|)$ berechnet, sowie die Knoten in dieser Reihenfolge ausgibt. Falls es keine topologische Sortierung gibt, soll `Der Graph ist nicht kreisfrei.` ausgegeben werden.

Sie bekommen von uns Funktionen zum Einlesen von Graphen und Datenstrukturen, die Graphen sowie die Daten des Algorithmus speichern können.

Wählen Sie geeignete Datenstrukturen und Methoden aus den vorgegebenen Methoden und implementieren Sie damit die topologische Sortierung. Verwenden Sie im Makefile `-D` um `LISTE` bzw. `MATRIX` zu definieren. Je nachdem, welche Wahl Sie für die Speicherung Ihres Graphen wählen.

Hinweise:

- Gehen Sie davon aus, dass die Knoten des Graphen durchgängig (beginnend mit 0) nummeriert sind. Ein Graph mit n Knoten hat also die Knoten $0, 1, \ldots, n-1$
- Wir starten den Algorithmus bei Knoten 0 und neue Zusammenhangskomponenten werden immer beginnend mit dem Knoten mit der kleinsten Nummer besucht.
- Während der Durchführung des Algorithmus werden Kanten in der Reihenfolge ihres Einlesens besucht.

Aufgabe 10.2: Labyrinth [10 Punkte, Link zur Abgabe] In Computerspielen ist eine häufige Problemstellung, dass eine Spielfigur (z.B. nach einem Mausklick des Benutzers) von ihrer aktuellen Position zu einer Zielposition bewegt werden soll. Dabei kann die Figur häufig nicht den direkten Weg nehmen, sondern wird durch Hindernisse blockiert.

Implementieren Sie mit den Mitteln der Vorlesung und der AlDat eine Wegfindungsmethode.

- Das Spielfeld ist ein Gitter und die Figur bewegt sich in einzelnen Schritten („Zügen“).
- Dabei kann die Figur nicht diagonal ziehen; vielmehr kann sie sich pro Zug nur um ein Feld nach links, rechts, oben oder unten bewegen.
- Start- und Zielposition im Gitter sind vorgegeben.
- Berechnen Sie, Züge mindestens benötigt werden, um von der Startposition zur Zielposition zu gelangen und geben Sie diese Anzahl aus.
- Kann die Zielposition von der Startposition aus nicht erreicht werden, geben Sie `-1` aus.
- Hindernissfelder dürfen nicht betreten werden. Ebenso darf das Spielfeld nicht verlassen werden.
- Wählen Sie geeignete Datenstrukturen aus dem Repo.

Listing 1: Beispiele

```
> cat example.in
++++++++++
+s  +   t+
+++ + ++++
  +   +
  +++++
> ./pathfinder < example.in
++++++++++
+s  +   t+
+++ + ++++
  +   +
  +++++
Von [1, 1] nach [8, 1] gelangt man mit 11 Schritten.
> ./pathfinder < example2.in
++++++++++
+s  +   t+
++++++++++
  +   +
  +++++
Von [1, 1] nach [8, 1] gelangt man mit -1 Schritten.
```

Für die volle Punktzahl finden Sie einen Algorithmus, der linear in der Anzahl der Gitterfelder ist.

Aufgabe 10.3: Clustering [10 Punkte, Link zur Abgabe] In dieser Aufgabe teilen wir zweidimensionale Punkte so auf k Cluster auf, dass die Cluster möglichst gut separiert sind, d.h., so dass der paarweise Abstand zweier Cluster maximal wird (der Abstand zweier Cluster A, B ist der kleinste Abstand eines Punktes aus A zu einem Punkt in B.)

Um das Clustering zu finden interpretieren wir die Punktmenge als einen Graphen, in dem es zwischen je zwei Punkten p, q aus der Eingabemenge eine Kante mit Länge $||p - q||^2$ gibt. Auf diesem Graphen führen wir den Algorithmus von Kruskal (s. Kapitel 8.5.3 im Aldat-Skript) aus: Fügt Kruskal eine Kante $\{p, q\}$ ein, bedeutet das, dass p und q im gleichen Cluster liegen.

Da Kruskal einen Spannbaum berechnet, würden am Ende des Algorithmus alle Knoten in einem einzigen, großen Cluster liegen. Wir stoppen den Algorithmus daher vorzeitig, wenn er genau k Cluster erzeugt hat.

Implementieren Sie diesen Algorithmus! Im Repository finden Sie alles Nötige.

Listing 2: Beispiel:

```
> cat example.in
4 2
1 0
2 0
1 10
2 10
> ./cluster < example.in
Cluster 1: 0 1
Cluster 2: 2 3
```

Die Eingabe enthält die Anzahl an Punkten n sowie die Anzahl k der gewünschten Cluster. Es folgen n Punkte. Die Ausgabe definiert k Cluster und gibt an, welcher Punkt in welchem Cluster liegt. Die Punkte werden dabei in der Reihenfolge der Eingabe, beginnend mit 0 nummeriert.

Der vorgegebene Code erzeugt aus der Eingabe bereits ein Array mit den Kanten des oben beschriebenen Graphen.

Hinweis: Falls Sie auf zwei Kanten mit der gleichen Länge stoßen, zählt die Kante als kürzer, deren Startknoten den kleineren Index hat. Sollten auch die Startindizes der beiden Kanten gleich sein, entscheidet der Index des Zielknotens, welche Kante kürzer ist.

Bemerkung: Dieser Algorithmus heißt *Single-Linkage*.