GeoMagic Testaufgabe

Filip Mazug

Kommentar

Im Folgenden möchte ich einen kurzen Überblick über den Algorithmus und das Konzept geben, damit das Einlesen leichter fällt. Aber zunächst noch ein kleiner Kommentar zur Aufgabenstellung. Ich hatte den Eindruck, der Satz: "Treffen sich mehr als zwei Linien in einem Punkt, dann wird an dieser Stelle nicht zusammengefügt.“, kann auf verschiedene Weisen interpretiert werden.

Option 1:
"In diesem Punkt wird nicht zusammengefügt", bedeutet nichts wird dort zusammengefügt. Somit gehen von diesem Punkt keine Linien aus.

Option 2:
Mehr als zwei Linien können in einem Punkt zusammenlaufen, aber in diesem Punkt werden dann die Linien nicht verbunden. D.h. alle Linien, die in diesem Punkt zusammenlaufen, enden dort. Oder fangen dort an, je nachdem wie man es betrachtet. Wenn sie dort anfangen, können sie von dort aus in andere Richtungen weiterlaufen. Diese Option bedeutet, dass in einem Punkt mehrere Linien quasi übereinander liegen können, die dort ihren Start oder Endpunkt haben, aber eben untereinander nicht verbunden sein können.

Option 3:
Die beiden Linien, die bereits mit einem Punkt verbunden sind, bleiben verbunden, aber keine weiteren Linien werden mit diesem Punkt verbunden, die hinterher dazukommen. In diesem Fall wäre die Reihenfolge entscheidend und man müsste alle Optionen betrachten.

Da die Aufgabenstellung die Zusammenhänge nicht näher spezifiziert, obwohl es für Option 1 und 2 eher nahe liegen würde dies zu tun, schien es mir am plausibelsten mich für Option 1 zu entscheiden. Die einfache Aussage der Aufgabenstellung trifft meiner Meinung nach am ehesten den Wortlaut von Option 1 und diese Option ist auch gleichzeitig diejenige, für die ich mich intuitiv als erstes entschieden habe, während des Erfassens der Aufgabenstellung.

Trotzdem wollte ich anmerken, dass mir andere mögliche Interpretationen bewusst sind und darauf eingehen. Je nachdem wie man das Ganze auslegt, wird dadurch auch bestimmt, wie der Algorithmus korrekt funktionieren muss und auch die Ergebnisse sind dadurch beeinflusst. Daher müsste man im Real Fall die Anforderungen und genauen Details entweder im Voraus definiert haben, oder sich mit dem Team und dem Teamleiter oder Kunden abstimmen, sobald die Multioptionalität auftaucht.

Das implementierte Konzept ließe sich jedoch auch auf Option 2 ummünzen unter Wiederverwendung der Komponenten und der generellen Idee. Einen grober Ansatz wird zur besseren Verständlichkeit am Ende skizziert.

Und nun zum Konzept:

- Eine einfache Datenstruktur Point speichert ein x,y Koordinatenpaar für besseres Handling
- Es gibt 3 mögliche Klassen von Points, welche durch die Aufgabenstellung vorgegeben sind.
    - SinglePoints (SP), die genau einmal vorkommen
    - Connectoren (CN), kommen genau zweimal vor und verbinden zwei Linien
    - Terminatoren (TER), kommen 3 oder mehr als dreimal vor.

- Da ich mich für Option 1 entschieden habe, also "nichts wird dort zusammengefügt, wo sich mehr als 2 Punkte Treffen", gehen von einem Terminator Punkt auch keine Linien aus. (Da ja auch keine hineingehen.)
- Das hat zur Folge, dass Linien, die Terminatoren enthalten, nicht für die Linienzüge verwendet werden. Das beruht darauf, dass eine Linie A, die einen TER enthält, für die Linienzüge per Definition nicht existiert, weil vom ersten Punkt (TER) (Anfang/Ende) dieser theoretischen Linie A, kein Strich zum zweiten Punkt (Anfang/Ende) der Linie A führt.
- Nachdem alle Linien entfernt wurden, die Terminatoren enthalten, bleiben die Linien übrig, die SPs und CNs enthalten.

- Ein Linienzug kann somit auch keine Verzweigung haben, da ein Connector höchstens 2 Partner haben kann und für eine Verzweigung mindestens 3 Partner gebraucht werden. Linienzüge können folgende Form aufweisen.
    - SP-SP, (einfache Linie)
    - SP-CN-SP, (Connector in der Mitte)
    - SP-CN-…(n x CN)...-CN-SP (mehrere Connectoren in der Mitte)
    - SP-CN gibt es nicht, da eine Linie nicht mit einem CN aufhören kann. Dieser braucht 2 Partner. Wenn er nur einen hat, ist es per Definition kein CN.
    - Stattdessen gibt es CN-CN oder CN-…(n x CN)…..-CN, was ein Kreis wäre. Dies löst der Algorithmus auch

- Auf diesen Überlegungen basiert der Algorithmus.
- Bevor ich den Ablauf erkläre, nochmal das Wichtigste zu den Datenstrukturen für die Klarheit:
    - singlePointsAndConnectors, enthält alle gültigen Punkte
      SPs haben type 1 und nach einmaliger Verwendung type 0
      CNs haben type 3 und nach einmaliger Verwendung type 2
      Nachdem ein CN das zweite Mal verwendet wurde, geht sein type von 2 auf 0 und type 0 markiert einen Punkt als gelöscht.
    - Ich habe mich für das Konzept mit den types entschieden, unter anderem um zu vermeiden, dass Listeneinträge aus Listen von Points gelöscht werden müssen, während man über diese Listen iteriert. Und weil die types eine praktische Möglichkeit bieten, um constrains für die Punkte zu speichern, während die Linienzüge gebildet werden.
    - StarterList, enthält alle gültigen Linien, dabei hat jeder Punkt in einer Linie type 1, der nach Verwendung auf type 0 geht, da, nachdem eine Linie verwendet wurde, diese nicht nochmal verwendet werden kann.
    - Durch einen Abgleich mit StarterList wird geprüft, ob die aktuell betrachtete Linie oder der Linienzug valide ist.
    - Durch SinglePointsAndConnectors und die zugehörigen Funktionen werden die wählbaren Pfade zur Verfügung gestellt. Durch die type Kontrolle wird die korrekte Auswahl von Punkten gewährleistet und wie häufig diese Punkte benutzt werden dürfen.

Vereinfachter Ablauf zur Orientierung

1. Basisfall: Wenn keine Punkte mehr übrig sind, um einen Linienzug zu bilden, beende die Rekursion und gib die aktualisierte Liste der Linienzüge zurück.
2. Nehme Point aus singlePointsAndConnectors.
3. Wenn der aktuelle Punkt ein SinglePoint ist (type == 1):
    - Suche nach einem zweiten SinglePoint oder Connector (CN hat Priorität), um eine Linie zu bilden.
    - Wenn ein zweiter Punkt gefunden wird (weil diese Kombination in starterList existiert), bilde Linie zwischen den beiden Punkten.
    - Wenn eine Linie erstellt werden kann:
        - Falls die Kombination (SP, SP) war: Füge die Linie zu den Linienzügen hinzu
            - Durch entsprechende Übergabe der Parameter erkennt die Funktion im rekursiven Aufruf, dass der Linienzug zu Ende war und ein neuer beginnt.
        - Falls der zweite Point ein CN war, muss der Linienzug weitergebaut werden: Füge die Linie zu Linienzug hinzu
            - Funktion erkennt im rekursiven Aufruf, dass der Linienzug weitergebaut werden muss.
        - Aktualisiere Point types (singlePointsAndConnectors und starterList)
        - Rufe die Funktion rekursiv auf, um den nächsten Linienzug zu erstellen, oder den aktuellen zu erweitern.
4. Wenn der aktuelle Punkt ein Connector ist (Type = 2 oder 3):
    - Suche nach einem weiteren Punkt, um den Linienzug fortzusetzen.
    - Solange ein weiterer Connector gefunden wird, setze den aktuellen Punkt auf den gefundenen Connector und such weiter.
        - Funktion erkennt im rekursiven Aufruf, dass der Linienzug weitergebaut werden muss
    - Wenn kein weiterer Connector gefunden wird, such nach einem abschließenden SinglePoint, um den Linienzug zu beenden.
        - Wenn ein abschließender SinglePoint gefunden wird, füg den Linienzug zu den Linienzügen hinzu.
    - Aktualisiere Point types (singlePointsAndConnectors und starterList)
    - Rufe die Funktion rekursiv auf, um den nächsten Linienzug zu erstellen, oder den aktuellen zu erweitern.
5. Rekursion: Teilmenge von verfügbaren Punkten (definiert durch die types) in SinglePointsAndConnectors wird weitergegeben sowie eine Teilmenge der noch nicht verwendeten Linien (starterList) während die Menge der Linienzüge wächst.
6. Rückgabe der Linienzüge: Am Ende der Rekursion wird die Liste der Linienzüge zurückgegeben, nachdem alle Punkte und Linien ordnungsgemäß verwendet wurden.

Kommentar

Mit der aufgeführten Idee und den dazugehörigen Komponenten ließe sich unter einiger Modifikation im Code auch die oben genannte Option 2 umsetzen. Man könnte zum Beispiel mit einem SP oder einem TER starten und den Linienzug nach dem gleichen Prinzip wachsen lassen, bis erneut entweder ein SP oder ein TER erreicht wird. Für Terminatoren müssten dafür neue types codiert werden, sodass diese mehrmals verwendet werden dürfen (je nachdem wie oft sie auftauchen als Punkt in einer Linie) und danach auf 0 gesetzt werden. In einem Terminator dürften nach Option 2 die Linienzüge anfangen oder enden, aber nicht über diesen „darüber gehen“. Somit ließen sich die Linienzüge mit ähnlicher Methodik aufbauen, was sich in die Logik des bisherigen Codes eingliedern würde.

Um Option 3 zu realisieren, müsste man die Reihenfolge der Verknüpfung beachten und an jedem Knoten alle Möglichkeiten durchprobieren und sich diese Möglichkeiten merken, um am Ende zu sehen, bei welchen Pfadkombinationen die meisten Linienzüge entstehen. Dabei würde man die Terminatoren und zugehörigen Linien nicht zu Anfang herausnehmen, sondern diese drin lassen, aber man dürfte sie nur einmal auf einem Linienzug verwenden. Danach wären sie nicht mehr wählbar. So würde durch jeden Terminator nur ein Pfad laufen. Eine Möglichkeit wäre den Algorithmus zufällig wählen zu lassen, welchen Pfad er nimmt. Und anschließend den Algorithmus sehr häufig aufrufen und mit einer hohen Iterationszahl sicherstellen, dass auch die Pfadkombinationen gefunden werden, welche die meisten Linienzüge ergeben (Für größere Probleme).