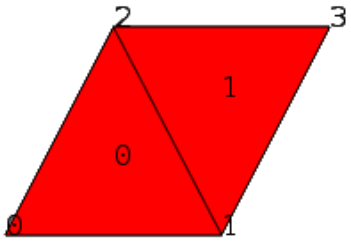
[DugJC] - 07 - Algorithmische Geometrie

Diese Challenge geht in Richtung Finite Elemente Methode (FEM) und Graphentheorie. FEM ist ein numerisches Verfahren zur Lösung von partiellen Differentialgleichungen. Dabei wird zunächst das Berechnungsgebiet in eine beliebig große Anzahl von Elementen unterteilt. Als Element wird in diesem Beispiel das Dreieck gewählt, weil es das einfachste geometrische Objekt im zwei dimensionalen Raum ist.

Ein Dreieck hat 3 Ecken (Knoten, Nodes). Jeweils 2 Ecken bilden eine Kante. Und 3 Kanten bilden das Dreieck. Verbindet man die Dreicke an ihren Kanten entsteht ein Netz. So können verschiedene Objekte nachgebildet werden. Die Nodes enthalten ihre Position im Raum oder der Fläche, also die X Y und Z Koordinaten und noch weitere Parameter. Wie Wassertiefe, Temperatur, Druck, Farbe.

Es soll in dieser Challenge nicht darum gehen, wie man daraus ein Gleichungssystem baut um mit den Verschiedenen Parameter zu rechnen. Es geht vielmehr um die Beziehungen zwischen Knoten, Kanten und Elemente. Z.B. kann ein Knoten die Ecke mehrere Dreiecke sein.

Hat man so ein Netz gegeben, möchte man es sich auch anzeigen oder andere Dinge mit tun. Die Knoten und Elemente werden durchnummeriert. Z.B.: Knoten 0, 1 und 2 gehöhren zu Dreieck 0. Und Knoten 1, 2 und 3 bilden Dreieck 1.



Hier ein Beispiel wie man das in C++ implementieren könnte.

```
class node
{
  // jeder Knoten bekommt eine Nummer
  int nodeNumber;
  // Die Koordinate im Raum
  double x, y;
  // Sein(e) zusaetzliche(n/r) Parameter
  double param;
};

class element
{
  // jedes Element bekommt eine Nummer
  int elementNumber;
  // Die 3 Knoten die das Element bilden
  node * knoten[3];
};
```

Soweit alles klar? Malt euch ein paar Punkte auf und verbindet sie zu Dreiecken. Ist ganz einfach.

Es gibt 4 Aufgaben, von denen ihr zwei lösen musst um 10 Punkte zu bekommen.

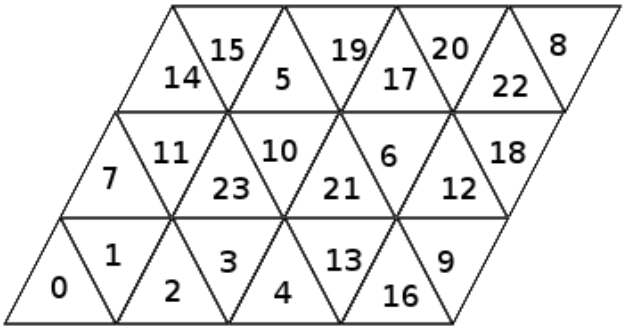
Aufgabe 1 und 2 kann getrennt von Aufgabe 3 und 4 gelöst werden.
Aufgabe 2 hängt von Aufgabe 1 ab.
Aufgabe 4 hängt von Aufgabe 3 ab.

Aufgabe 1: Nachbarn finden

Es liegen zwar Informationen vor, welche Knoten zu welchem Element gehören. Aber es gibt keine Informationen, welche Elemente über eine Kante miteinander verbunden sind. Die Aufgabe ist es einen Algorithmus zu finden, der die Nachbarn der Elemente ermittelt. Z.B. könnt ihr in der Klasse element eine neue Variable "int nachbar1, nachbar2 ..." anlegen, welche die Elementnummern der 3 Nachbarn enthalten.

Aufgabe 2: Netz sortieren

Nachdem ihr die Nachbar Informationen erstellt habt, könnt ihr das Netz neu sortieren. Sortiert werden sollen nur die Elemente, nicht die Knoten. Und zwar stell ich mir das so vor: Ihr fangt mit einem Element an, sagen wir das unterste. Das bekommt die Nummer 0. Eines seiner Nachbar Elemente bekommt dann die Nummer 1. Dessen Nachbar Element dann die Nummer 2 u.s.w. Man kann also mit aufsteigender Nummer von Element zu Element hüpfen.

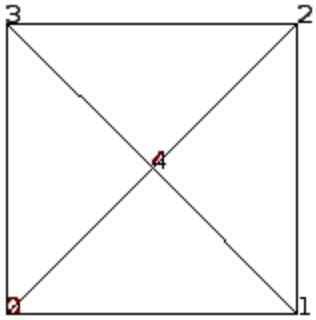


Die entscheidende Frage dabei ist, welches der 3 Nachbar Elemente gewählt wird. Das steht euch ganz frei. Vllt. habt ihr ja ein paar coole Ideen :) Man könnte immer das Element mit dem niedrigsten y-Wert nehmen. Oder Reih um alle Nachbarn.
In dem Bild sind die ersten 4 Elemente unten links schon sortiert. Der Rest vom Netz ist zufällig verteilt.

Das kann ganz schön kompliziert werden. Denkt etwas Objekt Orientiert. Vllt. eine element::naechsterNachbar() Funktion? Der Path von Element zu Element musst nicht zwangsläufig zusammenhängend sein. Denkt auch was aus :)

Aufgabe 3: Randknoten finden

Die Netze, die wir uns hier betrachten sind endlich und zwei dimensional. Das heißt, sie haben einen Rand. Findet heraus, wie man erkennen kann ob ein Knoten zum Rand gehört oder nicht. Löcher im Netz lassen sich auf die selbe Weise finden, da ein Loch ja auch ein Rand hat. Hier noch ein Tipp: man kann das nicht allein am Knoten erkennen. Es braucht schon die Informationen der angeschlossenen Elemente und/oder Kanten. Auch hier ist Objekt Orientiert schön. Vllt. eine Variable knoten::istRandKnoten ?



In dem Bild rechts sind Knoten 0 bis 3 Randknoten und Knoten 4 ist ein Gebietsknoten.

Aufgabe 4: Inseln und den Rand Polygon erstellen

Habt ihr erkannt was Rand und was Gebietsknoten ist, gilt es, sie in einer Reihenfolge (z.B. Gegen den Uhrzeigersinn) herauszuschreiben. Die Reihenfolge ist wichtig, so dass dann ein Polygon entsteht. Ihr müsst also irgendwie herausfinden wie die Randknoten verbunden sind.

Hilfsmittel & sonstige Bedingungen

Weil nur zwei Wochen zur Verfügung stehen, ist es nicht Teil der Aufgabe ein Programm zu schreiben welches die Elemente graphisch anzeigt. Es ist bei so kleine Netze auch nicht nötig. Wenn ihr Lust habt, könnt ihr mein Programm nutzen um euch die Elemente und Knoten anzuzeigen. Es ist in Qt geschrieben. Entweder die Binaries probieren und wenn das nicht klappt: selbst compilieren :)
Das Programm mit Screenshots findet ihr unter http://sourceforge.net/projects/katerfempresent/
Sollte es Probleme damit geben, oder habt ihr Ideen was man verbessen könnte, schreibt es doch gerade ins Forum. Auch dürft ihr jederzeit zu den Aufgaben Fragen stellen.

Ihr dürft jede Sprache nutzen die ihr wollt. Sollen sehr exotische Sprachen dabei sein, bitte ich darum den Code gut zu kommentieren, da ich nur C++ und verwandte Sprachen kann. Und solltet ihr euch komplexe Algorithmen ausdenken, wäre eine Beschreibung nett, so dass ich nachvollziehen kann was passiert. Ihr könnt euch auch gern ein Algorithmus googlen und dann selbst umsetzen.
Ziel dieser Aufgabe ist es, dass ihr Spaß dran findet. Und mal selbst knobeln macht doch mehr Spaß oder? ;)

Anbei sind ein paar Testnetze. Das Format der Datei ist ganz einfach, so dass ihr euch selbst Netze schreiben könnt wenn ihr wollt. Und das Parsen der Datei ist dann auch nicht so schwer. Hier ein Beispiel für zwei Dreiecke:

```
C Anzahl der Randknoten:
       0
C Anzahl der Gebietsknoten:
       4
C Koordinaten und Skalarwerte der Knoten
C -------------------------------------
C Zuerst die Randknoten  (Anzahl s.o.),
C dann die Gebietsknoten (Anzahl s.o.).
C ------------+-------------+-------------+---------------
C     Nr.     |  x-Koord.   |   y-Koord.  |   Skalarwert
C ------------+-------------+-------------+---------------
      0            0               0             0
      1           10               0             0
      2            5              10             0
      3           15              10             0
C ------------------------------------------------------------
C Anzahl der Elemente:
      2
C Elementverzeichnis
C ------------------------------------------------------------
C    Knoten i  Knoten j  Knoten k   Kennung     Nr.
        0         1          2          0          0
        1         2          3          0          1
C ------------------------------------------------------------
```

Erklärung:
Der Aufbau einer Datei ist immer gleich. Kommentare werden mit dem Zeichen “C” eingeleitet. Dann kommt die Anzahl der Rand und Gebietsknoten. Da keine Randknoten gegeben sind, stehen in den Beispielen hier immer eine 0. Die Knoten werden Zeilenweise angegeben. Erst eine vorlaufende Nummer, dann X und Y Koordinate und zuletzt ein Parameter. Der Parameter ist unwichtig, ihr könnt ihn also ignorieren.
Danach wird die Anzahl der Elemente angegeben und schließlich die Elemente selbst. Erst kommen die drei Knotennummern, aus denen das Dreieck besteht. Dann eine Kennung, welche ihr auch ignorieren könnt und am Ende eine vorlaufende Nummer.

Bewertung

Wie gesagt müßt ihr zwei Aufgaben bearbeiten um 10 Punkte zu bekommen. Jede Aufgabe hat also 5 Punkte. Und ich unterteile nochmal zwischen Theorie und Praxis. Wer sich ins stille Kämmerlein stetzt und 10 Algos ausbrütet aber davon keinen umsetzt, bekommt genauso viele Punkte wie jemand der mir 10 Seiten Code schickt, der auf keinerlei Theorie aufbaut.

Zusatzaufgabe

In der Klasse element werden die drei Knoten vom Dreieck als drei Pointer vom Typ node gespeichert. Also das Element hat ein Pointer auf seine Nodes. Satt ein Pointer könnte man auch einfach die Knotennummer speicher. Was für Vor- Nachteile hat das? Laufzeit, Speicherverbrauch, Sicherheit, Programmierkomfort.

Beispiel: sortieres Netz

Link unsortiert. Rechts sortiert. Die Farbe zeigt die Elementnummer. Von rot über orange, gelb, grün, blau, rosa und wieder rot.