Programming C++

Project **Rundreiseproblem**

Status: 25. Mai 2021, 19:36

Supervisor: Prof.Dr. G. Haase, `gundolf.haase@uni-graz.at`

Implementieren Sie drei Algorithmen, die das Rundreiseproblem (TSP) lösen (siehe dazu Wikipedia[1]) und werten Sie die einzelnen Verfahren statistisch aus. (8 Pkt.)

- Schreiben Sie eine Funktion, die die Datei Graph_6.txt[2] einlesen kann.

```
6
1 0 2
2 2 2
3 0 0
4 2 0
5 1 1
6 4 0
```

  Die erste Zeile beinhaltet die Anzahl der Knoten und alle weiteren Zeilen beinhalten den Index, die x- und die y-Koordinate jedes einzelnen Knotens. Die Kantengewichte entsprechen den Euklidischen Distanzen. Den in dieser Datei gespeicherten Graphen finden Sie in Abbildung 1.

- Definieren Sie eine Klasse `Graph`, die einen symmetrischen gewichteten Graphen speichert. Schreiben Sie auch einen Ausgabeoperator (`operator<<`).
- Definieren Sie eine Funktion `laenge`, die die Länge einer Tour, die als `vector` gespeichert ist, berechnet. Überprüfen Sie, dass die Tour $1 \to 4 \to 6 \to 2 \to 3 \to 1$ die Länge $4 + 6\sqrt{2} \approx 12,49$ hat.
- Definieren Sie nun eine abstrakte Klasse `Algorithmus`, die als Member die Lösung (in unserem Fall die erhaltene Tour), den Zielfunktionswert (die Länge der Tour) und die Laufzeit beinhaltet. Außerdem sollte in dieser Klasse eine Methode `loese(const Graph &graph)` implementiert sein, die eine abstrakte Methode `loesungsalgorithmus` aufruft und für die richtige Bemessung der Laufzeit sorgt. (Hinweis: Die Methode `loesungsalgorithmus` soll dabei nicht implementiert sein!)
- Implementieren Sie nun eine Klasse `NearestNeighbour`, die von der Klasse `Algorithmus` abgeleitet ist, die Methode `loesungsalgorithmus` implementiert und eine Tour mithilfe der Nearest-Neighbour-Heuristik berechnet (siehe dazu Wikipedia[3]). Der Startknoten wird über den Konstruktor übergeben. Überlegen Sie

[1] `http://de.wikipedia.org/wiki/Problem_des_Handlungsreisenden`

[2] `http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS21/projects/Rundreiseproblem/Graph_6.txt`

[3] `http://de.wikipedia.org/wiki/Nearest-Neighbor-Heuristik`

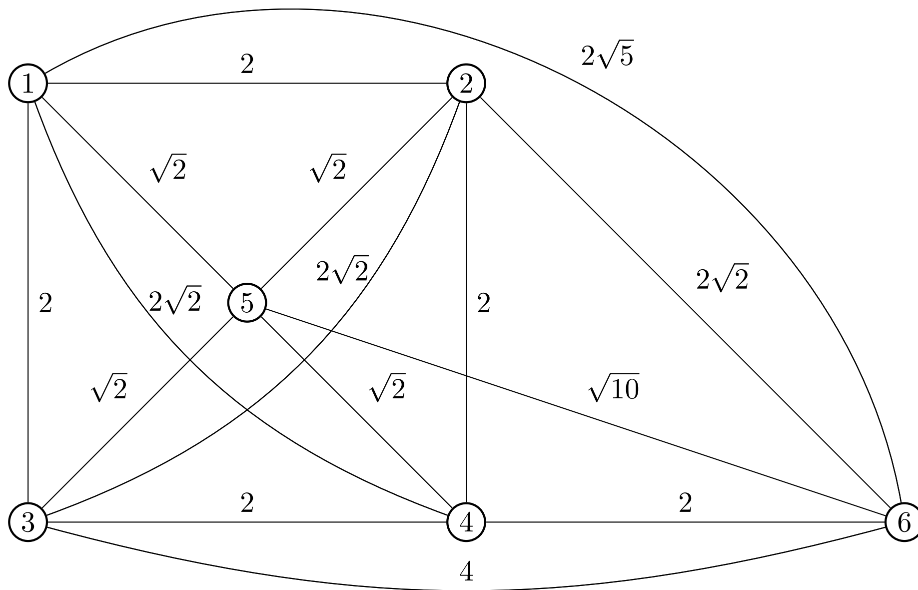

Abbildung 1: Graph

dabei, welche Container und Funktionen für diese Aufgabe geeignet sind (schauen Sie z.B. die Funktionen `sort`[4], `min`[5] und `min_element`[6] an).

- Ähnlich zu der Nearest-Neighbour-Heuristik implementieren Sie nun die Nearest-Insertion-Heuristik in Form einer neuen Klasse `NearestInsertion` (siehe dazu Wikipedia[7]).
- Anschließend implementieren Sie eine Klasse `Exakt`, die eine minimale Tour durch das Ausprobieren aller möglichen Touren findet. (Eine der Möglichkeiten zum Lösen der Aufgabe bietet der Algorithmus von Heap[8] an.)
- Schreiben Sie nun ein Hauptprogramm, das beide Lösungsverfahren (die Nearest-Neighbour- und Nearest-Insertion-Heuristik) für alle Instanzen, Graph_6.txt[9],

[4] `http://www.cplusplus.com/reference/algorithm/sort/`

[5] `http://www.cplusplus.com/reference/algorithm/min/`

[6] `http://www.cplusplus.com/reference/algorithm/min_element/`

[7] `https://de.wikipedia.org/wiki/Nearest-Insertion-Heuristik`

[8] `http://en.wikipedia.org/wiki/Heap's_algorithm`

[9] `http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS21/projects/Rundreiseproblem/Graph_6.txt`

Graph_7.txt[10], Graph_8.txt[11], Graph_9.txt[12], Graph_10.txt[13], Graph_11.txt[14] und Graph_12.txt[15], testet und mit der optimalen Lösung vergleicht. Benützen Sie dabei einen Vektor vector<Algorithmus *>, in dem alle zu testende Verfahren gesammelt sind. Für jedes Verfahren und für jede Instanz berechnen Sie die Güte als *heuristische Lösung / optimale Lösung* und anschließen bestimmen Sie für jeden Algorithmus eine durchschnittliche Güte in Form eines (geometrischen) Mittelwertes der einzelnen Güte. Welches von den beiden Verfahren ist besser? Und welches ist schneller?

Zusatzaufg.: Definieren Sie nun eine weitere Klasse `Opt2`, die die 2-Opt-Heuristik (siehe dazu Wikipedia[16]) implementiert. Das Startverfahren (in unserem Fall die Nearest-Neighbour- oder Nearest-Insertion-Heuristik) wird dabei als Member der Klasse `Opt2` deklariert und über den Konstruktor übergeben. (+1,5 Pkt.)

Zusatzaufg.: Implementieren Sie die Lin-Kernighan-Heuristik als ein weiteres Verbesserungsverfahren (siehe dazu folgende Seminararbeit[17]). Das Startverfahren (in unserem Fall die Nearest-Neighbour- oder Nearest-Insertion-Heuristik oder aber auch die 2-Opt-Heuristik) wird dabei ebenfalls als Member der Klasse deklariert und über den Konstruktor übergeben. (+2,5 Pkt.)

Zusatzaufg.: Betrachten Sie nun eine andere Problemstellung: Sie suchen ebenfalls eine Tour (Sie wollen also wieder alle Knoten genau einmal besuchen), aber anstatt der Gesamtdistanz wollen Sie nun die Summe der inneren Winkel (+2 Pkt.)

$$\alpha_{ijk} := \pi - \arccos_{[0,\pi]}\left(\frac{j-i}{||j-i||} \cdot \frac{k-j}{||k-j||}\right)$$

(siehe auch Abbildung 2) in der Tour minimieren. Schreiben Sie eine Funktion `winkellaegne`, die die Summe der inneren Winkel für eine gegebene Tour berechnet. Ändern Sie nun alle Ihre Algorithmen-Klassen so, dass die Funktion, die den Zielfunktionswert (in unserem Fall also die Länge oder die Summe der inneren Winkel) berechnet, als Parameter im Konstruktor übergeben wird. Beachten Sie dabei, dass die ganze **Funktion als Parameter** übergeben wird, und dass die Verfahren also auch für andere Zielfunktionswert-Funktionen ohne Änderungen im Quellcode durchführbar sein müssen! Bestimmen Sie anschließend die Güte aller implementierten Verfahren auch mit der Funktion `winkellaenge`.

[10] http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS21/projects/Rundreiseproblem/Graph_7.txt
[11] http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS21/projects/Rundreiseproblem/Graph_8.txt
[12] http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS21/projects/Rundreiseproblem/Graph_9.txt
[13] http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS21/projects/Rundreiseproblem/Graph_10.txt
[14] http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS21/projects/Rundreiseproblem/Graph_11.txt
[15] http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS21/projects/Rundreiseproblem/Graph_12.txt
[16] https://de.wikipedia.org/wiki/K-Opt-Heuristik
[17] https://tobi.rocks/pdf/seminararbeit_2013.pdf

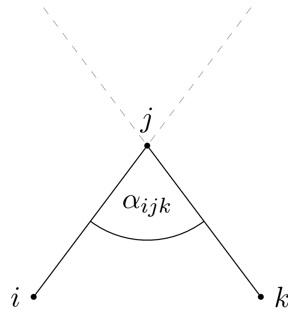

Abbildung 2: Innerer Winkel α_{ijk}.

Hinweis: Wer sich für das Problem des Handlungsreisenden interessiert, kann das Concorde[18]-Projekt anschauen.

[18] http://www.math.uwaterloo.ca/tsp/concorde.html