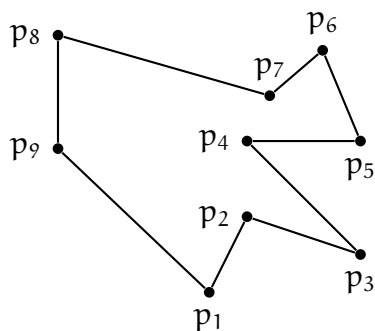


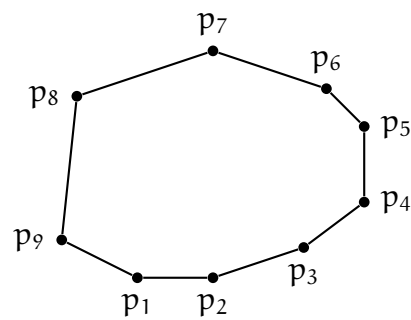
3. Übungsblatt zur Vorlesung Algorithmische Graphentheorie (Sommer 2022)

Aufgabe 1 – Triangulierungen und Dynamische Programmierung

Ein *Polygon* ist ein geschlossener Streckenzug in der Ebene – eine Folge von n verbundenen Punkten $p_1, p_2, \dots, p_n$, die wieder am Startpunkt p_1 endet. Die Punkte $p_1, p_2, \dots, p_n$ des Polygons werden *Ecken* genannt und die Strecken zwischen den Ecken *Polygonkanten*. Ein Polygon ist *konvex*, wenn jede Strecke mit Start- und Endpunkt im Polygon auch selbst vollständig im Polygon enthalten ist; siehe Abbildung 1.



(A) nicht-konvexes Polygon

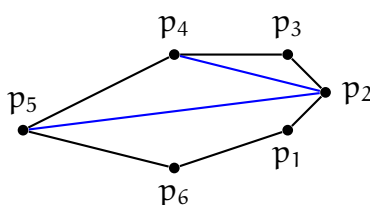


(B) konvexes Polygon

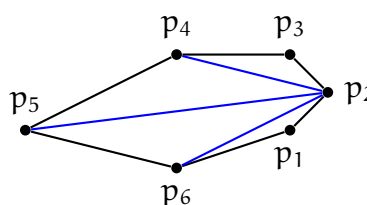
ABBILDUNG 1: Beispiele für Polygone.

Eine *Triangulierung* eines Polygons ist eine nicht-erweiterbare Menge von Strecken, die nicht-benachbarte Ecken verbinden, im Inneren des Polygons liegen, keine Polygonkanten schneiden und sich nicht gegenseitig schneiden. Wir nennen diese Strecken *Diagonalen*.

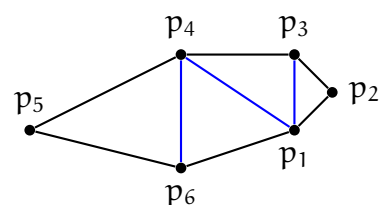
Wenn wir ein konvexes Polygon triangulieren, so müssen wir offensichtlich nur darauf achten, dass sich die Diagonalen nicht gegenseitig schneiden. Die *Kosten* einer Triangulierung sind die Summe der euklidischen Längen aller ihrer Diagonalen. Entsprechend heißt die Triangulierung mit den geringsten Kosten *kostenminimal*; siehe Abbildung 2.



(A) keine Triangulierung



(B) Triangulierung, die nicht kostenminimal ist



(C) kostenminimale Triangulierung

ABBILDUNG 2: Triangulierungen eines konvexen Polygons. Die Diagonalen sind blau gefärbt.

Wir entwickeln nun schrittweise ein dynamisches Programm, um die Kosten einer kostenminimalen Triangulierung eines gegebenen konvexen Polygons P zu bestimmen.

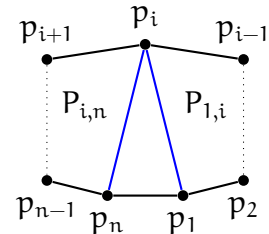
Sie können annehmen, dass wir die Länge $d(p_i, p_j)$ einer Diagonalen (p_i, p_j) in konstanter Zeit bestimmen können und dass es keine drei Ecken des Polygons gibt, die auf einer Geraden liegen.

- a) In einer kostenminimalen Triangulierung von P ist die Polygonkante (p_1, p_n) Teil eines Dreiecks $\Delta p_1 p_i p_n$, wobei $i \in \{2, \dots, n-1\}$. Das Dreieck $\Delta p_1 p_i p_n$ trennt von P zwei Teilpolygone $P_{1,i}$ und $P_{i,n}$ ab.

Angenommen wir würden die Kosten für eine kostenminimale Triangulierung von $P_{1,i}$ und $P_{i,n}$ kennen, wie könnten wir daraus die Kosten einer kostenminimalen Triangulierung von P berechnen?

Beachten Sie auch die Spezialfälle $i = 2$ und $i = n - 1$.

2 Punkte



- b) Beschreiben Sie ein dynamisches Programm in Worten, welches für ein gegebenes konvexes Polygon die Kosten einer kostenminimalen Triangulierung berechnet.

Das dynamische Programm soll polynomielle Laufzeit haben und eine Tabelle $A[\cdot, \cdot]$ verwenden, welche für jedes Teilpolygon $P_{i,j}$ von P mit Ecken $p_i, p_{i+1}, \dots, p_{j-1}, p_j$ und $1 \leq i < j \leq n$ einen Eintrag $A[i, j]$ für die Kosten einer kostenminimalen Triangulierung enthält.

4 Punkte

- c) Geben Sie Ihren Algorithmus aus Teilaufgabe b) in Pseudocode an.

3 Punkte

- d) Geben Sie scharfe obere Schranken für die asymptotische Laufzeit und den asymptotischen Speicherbedarf Ihres dynamischen Programms an.

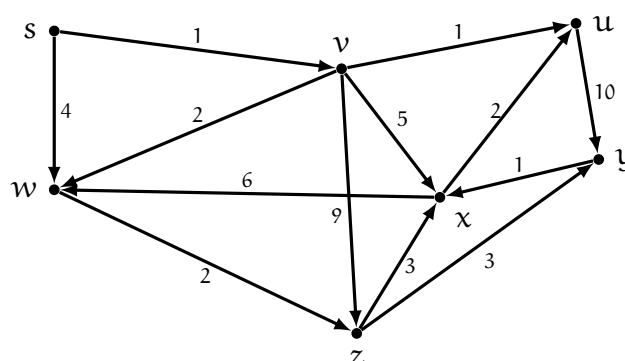
1 Punkt

Aufgabe 2 – Algorithmus von Dijkstra

Ein grundlegendes Problem in der algorithmischen Graphentheorie ist die Suche nach kürzesten Wegen. Anwendung findet dies beispielsweise in der Routenplanung, bei der wir typischerweise einen gerichteten Graphen mit Kantengewichten vorliegen haben. Zum Suchen von kürzesten Wegen eignet sich hier besonders der Algorithmus von Dijkstra. Diesen haben wir bereits in der Vorlesung Algorithmen und Datenstrukturen kennengelernt und in den Wiederholungsfolien noch einmal aufgegriffen.

Ermitteln Sie für den abgebildeten Graphen mit Hilfe des Algorithmus von Dijkstra einen Baum kürzester Wege bezüglich s . Geben Sie in nachvollziehbarer Art und Weise die Zwischenschritte an.

4 Punkte



Aufgabe 3 – Metrisches TSP

In der Vorlesung wurde ein Approximationsalgorithmus auf Basis von Spannbäumen für das metrische TSP vorgestellt. Dieser besitzt *Güte 2*, d.h. die Rundreise, die der Algorithmus liefert, ist höchstens doppelt so lang wie eine optimale Rundreise.

Im Folgenden sei G ein vollständiger, metrischer, ungerichteter Graph mit Kantengewichten $c: E \rightarrow \mathbb{R}_{\geq 0}$. Betrachten Sie den folgenden Algorithmus:

CompleteHamilton(Graph $G = (V, E)$, Kostenfunktion c)

 wähle $s \in V$ beliebig

 wähle eine Kante $\{s, t\} \in E$ mit $c(\{s, t\})$ minimal

$C \leftarrow \langle \{s, t\}, \{t, s\} \rangle$

while $V(C) \neq V$ **do**

 wähle einen Knoten $v \in V \setminus V(C)$, dessen Abstand zu C am geringsten ist

 sei w ein Knoten in C mit $c(\{v, w\})$ minimal

 sei u einer der beiden Nachbarn von w auf C

 ersetze in C die Kante $\{u, w\}$ durch die Kantenfolge $\langle \{u, v\}, \{v, w\} \rangle$

return C

- a) Zeigen Sie, dass der Algorithmus von Prim (beginnend mit Startknoten s) in jedem Schritt denselben Knoten zum aktuellen Baum hinzufügt wie Algorithmus CompleteHamilton zum Kreis C . **2 Punkte**

Hinweis: Sie dürfen vernachlässigen, dass es in einem Schritt mehrere gleich gute Wahlmöglichkeiten geben kann.

- b) Zeigen Sie, dass der Algorithmus CompleteHamilton Güte 2 besitzt. **4 Punkte**

Hinweis: Verwenden Sie Teilaufgabe a) sowie die *Dreiecksungleichung*.

Bitte laden Sie Ihre Lösungen als pdf bis **Dienstag, 24. Mai 2022, 13:00 Uhr** im WueCampus-Kursraum beim 3. Übungsblatt hoch. Geben Sie stets die Namen aller an, die das Übungsblatt bearbeitet haben (max. 2).

Begründen Sie Ihre Behauptungen und kommentieren Sie Ihren Pseudocode!