

3. Übungsblatt

Jasper Gude Pia Röttgers

11. Mai 2026

22 / 25

Aufgabe 1 – Triangulierungen und Dynamische Programmierung

a) ✓

$$T(P) = \begin{cases} 0 & \text{falls } P \text{ nur 2 Ecken hat} \\ \min_{i \in \{2, \dots, n-1\}} [T(p_{1,i}) + T(p_{i,n}) \\ + \text{Diagonalkosten}(i)] & \text{sonst} \end{cases}$$

Das versucht schon mehr von der b) zu lösen, als die a) fragt.

$$\text{Diagonalkosten}(i) = \begin{cases} d(p_2, p_n) & \text{falls } i = 2 \\ d(p_1, p_{n-1}) & \text{falls } i = n - 1 \\ d(p_1, p_i) + d(p_i, p_n) & \text{sonst} \end{cases}$$

Die Kosten der minimalen Triangulierung $T(P)$ lassen sich aus der minimalsten Summe der kostenminimalsten Triangulierung der entstehenden Teilpolygone $T(p_{1,i})$ und $T(p_{i,n})$ und den entstehenden Diagonalkosten berechnen.

Die Diagonalkosten unterscheiden sich, je nachdem ob eine oder zwei der drei Kanten Polygonkanten sind. Sobald ein (Teil-)Polygon nur noch zwei Ecken hat bricht die Rekursion ab.

2 / 2

b) Man nutzt die Idee aus a). ✓

Tabelle $A[i, j]$ speichert die kostenminimalen Triangulierung des Teilpolygons mit den Ecken p_i, \dots, p_j .

Für alle i gilt $A[i, i + 1] = 0$, da diese Teilpolygone mit nur zwei Ecken darstellen, welche keine Triangulierung benötigen.

b) perfekt.

Vor allem die markierte Aussage wird sehr gerne vergessen. Deswegen markier ich sie nochmal.

Man muss Fehler nicht selbst machen, um aus Fehlern zu lernen. =)

Für $j > i + 1$ gilt:

$$A[i, j] = \min_{k \in \{i+1, \dots, j-1\}} [A[i, k] + A[k, j] + \text{Diagonalkosten}(i, k, j)]$$

$$\text{Diagonalkosten}(i, k, j) = \begin{cases} d(p_k, p_j) & \text{falls } k = i + 1 \\ d(p_i, p_k) & \text{falls } k = j - 1 \\ d(p_i, p_k) + d(p_k, p_j) & \text{sonst} \end{cases}$$

Man berechnet dabei die Einträge nach nach wachsendem Abstand $r = j - i$, also von $r = 2$ bis $r = n - 1$. Dadurch sind die Einträge $A[i, k]$ und $A[k, j]$ immer bereits berechnet wenn man den $A[i, j]$ benötigt, da $k - i < r$ und $j - k < r$ gilt.

Das Ergebnis steht dann in $A[1, n]$.

4 / 4

- c) Die polynomielle Laufzeit kann über die Schleifen des DP begründet werden.

Äußere Schleife: r läuft von 1 bis $n - 1$ benötigt also $\mathcal{O}(n)$ Schritte.

Mittlere Schleife: für jedes r gibt es $\mathcal{O}(n)$ Paare (i, j) bei welchen $j - i = r$ gilt.

Innere Schleife: Für jedes Paar probiert man alle k zwischen i und j also $\mathcal{O}(n)$ Werte.

Pro Iteration der inneren Schleife benötigt man $\mathcal{O}(1)$ um die Einträge zu berechnen.

Daraus folgt:

Was sind die 3 Schleifen?

In eurem Algorithmus sind keine Schleifen benannt.

Die Erklärung unten über Tabelleneinträge ist mir alleine schon genug, aber mit dem Abschnitt über diesem Kommentar kann ich wenig anfangen.

$$T(n) = \mathcal{O}(n) \cdot \mathcal{O}(n) \cdot \mathcal{O}(n) \cdot \mathcal{O}(1) = \mathcal{O}(n^3)$$

Diese Schranke ist auch scharf, da jeder der $\mathcal{O}(n^2)$ Tabelleneinträgen auch tatsächlich befüllt wird und für jeden der Einträge im Durchschnitt $\mathcal{O}(n)$ Werte von k durchprobiert werden. Es gilt also $\Theta(n^3)$.

Der Speicherbedarf des Programms ist $\mathcal{O}(n^2)$, da die Tabelle so viele Einträge für die Paare (i, j) mit $1 \leq i < j \leq n$ hat.

2 / 2

Aufgabe 2 – TSP mit Wiederholungen

- a) Um TSP mit Wiederholungen auf Metrisches TSP zu reduzieren, müssen wir den zugrundeliegenden Graphen metrisch machen, d. h. die Dreiecksungleichung muss für jede Menge von 3 Knoten gelten.

Die Aussage hat in nahezu allen Abgaben gefehlt. Wundervoll!

Dazu iterieren wir über alle möglichen Mengen $T = \{a, b, c\}$ mit $a \neq b \neq c$ und $a, b, c \in V$ und $G(T, E)$ ist vollständig. Erfüllt eine Menge die Dreiecksungleichung $c(a, b) \leq c(b, c) + c(a, c)$ nicht, so löschen wir

2

Die Idee ist ganz cool und würde das Problem sogar lösen. Aber was ihr tut ist keine Reduktion auf Metrisches TSP.

Gesucht ist eine Übertragung eures Ausgangsproblems auf einen Graphen, der den Anforderungen eines metrischen TSP entspricht. Die "metrische" Komponente davon erfüllt euer Ansatz, allerdings fordert TSP einen vollständigen Graphen, ihr baut einen Baum. Für den Baum funktioniert zwar unser Approximations-Ansatz noch, aber das metrische TSP selbst scheitert. Es sucht die kürzeste Rundreise, im Baum gibt es keine Rundreisen.

Da das die erste Übungsaufgabe zu Reduktion ist, korrigiere ich freundlich, und da ihr das Problem trotzdem löst, alle Schritte sehr sauber begründet und angibt was für "metrisch" erfüllt sein muss, gebe ich noch 2.5/3.

die Kante mit den höchsten Kosten aus dem Graphen. Das dürfen wir, da die TSP-Tour diese Kante nie enthalten wird, weil es einen Weg gibt, der kürzer ist und alle drei Knoten enthält. Der Graph bleibt zusammenhängend, da wir für jede Menge T nur eine Kante löschen.

2.5 / 3

- b) Da wir den Graphen jetzt auf einen metrischen reduziert haben, können wir ähnlich wie in der Vorlesung vorgehen.

Dazu nehmen wir einen Minimalen Spannbaum des Metrischen Graphen. Durch verdoppeln der Kanten entsteht ein Kreis für dessen Kosten gilt (siehe Vorlesung):

$$c(\text{Kreis}) = 2 \cdot c(\text{MSB}) \leq 2 \cdot \text{OPT}$$

Da eine TSP-Tour mit einer Kante weniger ein Spannbaum ist.

Da wir Knoten und Kanten mehrfach benutzen dürfen, ist dieser Kreis eine 2-Approximation für TSP mit Wiederholungen.

3 / 3

Aufgabe 3 – Metrisches TSP

- ✓ a) Der Algorithmus für Minimale Spannbäume von Prim fügt in jedem Schritt die den Schnitt kreuzende Kante zum Baum hinzu, deren Kosten unter allen anderen kreuzenden Kanten minimal ist.

COMPLETEHAMILTON fügt denselben Knoten hinzu, da der Knoten v nicht in C liegt, also den Schnitt kreuzt und den kleinsten Abstand zu den Knoten in C hat.

2 / 2

- b) In jedem Schritt von COMPLETEHAMILTON wird eine Kante wu in C durch zwei Kanten wv und vu ersetzt. Da wv die minimale kreuzende Kante ist und G die Dreiecksungleichung erfüllt, ist $c(v, u) \leq c(u, w) + c(w, v)$ und ist somit nicht länger als der doppelte Minimale Spannbaum, der analog zu COMPLETEHAMILTON berechnet wird. Damit gilt wieder:

$$c(\text{Kreis}) = 2 \cdot c(\text{MSB}) \leq 2 \cdot \text{OPT}$$

Da eine TSP-Tour mit einer Kante weniger ein Spannbaum ist.

2.5 / 4

Für die markierte Aussage wäre weitere Umformung oder Argumentation hilfreich, um die Folgerung zu stützen, dass C eine 2-Approx des MSB bleibt. (Ihr schreibt, was die Kante $c(v,u)$ kleiner ist als der MSB, was etwas skurril ist). Ansonsten sehr solider Induktionsschritt... der leider seinen Induktionsanfang vermisst.

Aufgabe 4 – Längste Wege

- a) Da s, t in G' adjazent zu jedem Knoten in G ist, können wir einen einfachen s - t -Weg der Länge $k + 2$ erzeugen, indem wir einen einfachen Weg der Länge k in G nehmen, s an das eine Ende und t an das andere Ende hängen.

Umgekehrt kann man aus einem einfachen s - t -Weg der Länge k in G' einen einfachen Weg der Länge $k - 2$ in G konstruieren, indem wir s und t entfernen.

2 / 2

- b) Ein Hamiltonweg ist ein Weg der alle Knoten in G beinhaltet und somit Länge $n - 1$ besitzt.

Wie wir oben gezeigt haben, kann ein s-t-Weg der Länge $n + 1$ in G' leicht in einen Weg der Länge $n - 1$ in G umgewandelt werden. Das heißt, dass wir einen Hamiltonweg in G finden, wenn wir einen s-t-Weg finden.

Umgekehrt können wir einen Hamiltonweg leicht in einen s-t-Weg umwandeln, also finden wir einen s-t-Weg wenn wir einen Hamiltonweg finden.

Also finden wir einen Hamiltonweg genau dann, wenn wir einen s-t-Weg finden.

In der Aussage fehlt mir eine Angabe der Länge des s-t-Weges. Der Rest der Antwort ist klar genug und 1 Punkt ist wenig Spielraum.

Aber so wie es da steht, ist die Implikation
s-t-Weg gefunden \Rightarrow Hamiltonweg
nicht korrekt. Einen s-t-Weg finden wir laut Aufgabenstellung immer, solange der Ausgangsgraph Knoten enthält.

1 / 1

- c) Da wir Hamiltonweg auf LÄNGSTER s-t-WEG reduziert haben, muss also LÄNGSTER s-t-WEG NP-schwer sein, denn wenn es in P liegen würde, könnten wir auch Hamiltonweg in polynomialer Zeit lösen. Da wir nicht von $P = NP$ ausgehen, ist das nicht möglich.

1 / 2

Auch wenn es in diesem Fall sehr offensichtlich wirkt:
für eine vollständige Argumentation einer Polynomialzeit-Reduktion muss explizit angesprochen werden, dass die Übersetzung von einem Problem auf das andere in Polynomialzeit möglich ist.
Ohne diese Information besteht noch kein Widerspruch zwischen "Hamiltonweg ist NP-schwer" und "s-t-Weg ist in Polynomialzeit lösbar".

Nicht jeder längste s-t-Weg entspricht einem Hamilton-Weg. Mir fehlt hier noch ein Schritt.