Technische Universität München
Zentrum Mathematik
Prof. Dr. P. Gritzmann, Dipl.-Inf. Dipl.-Math. S. Borgwardt, Dr. M. Ritter

Optimierung 2, WS 2008/09

Übungsblatt 10

Aufgabe 10.1
Betrachten Sie das Problem SUBSET SUM aus Aufgabe 8.2:

Problem (SUBSET SUM)
Gegeben: *Eine Menge* $S = \{s_1, ..., s_n\}$ *mit* $s_i \in \mathbb{Z}\ \forall i \in \{1, ..., n\}$, $\beta \in \mathbb{Z}$.

Auftrag: *Entscheide, ob eine Teilmenge* $S' \subset S$ *existiert, so dass*

$$\sum_{x \in S'} x = \beta.$$

a) Zeigen Sie, dass SUBSET SUM in $\mathcal{NP}$ liegt.

b) Geben Sie einen Algorithmus an, der SUBSET SUM mit dynamischer Optimierung löst und dazu eine *Rückwärtsrekursion* verwendet. Beachten Sie den Sonderfall $\beta = 0$!

c) Beweisen Sie, dass SUBSET SUM (durch dynamische Optimierung) in pseudo-polynomieller Zeit gelöst werden kann.

Lösung zu Aufgabe 10.1

a) Als Zertifikat benutzen wir eine gegebene Teilmenge $S' \subset S$. Wir können in polynomieller Zeit testen, ob die gegebene Teilmenge sich auf β aufsummiert und damit eine Lösung für SUBSET SUM ist.

b) Bei einer Rückwärtsrekursion bauen wir unseren Entscheidungsschichtgraphen "von hinten" auf. Wir konstruieren ihn ähnlich wie in der Vorlesung: Jedes Element $s_i \in S$ induziert eine Schicht V_{i-1} von Knoten. Die Kanten des Graphen verlaufen jeweils zwischen Schichten V_{i-1}, V_i und entsprechen den Entscheidungen, ob s_i für S' benutzt wird oder nicht. Die Knoten in Schicht V_i bezeichnen wir mit $v_{i,j}$, dabei entspricht j der in den weiteren Schritten notwendigen Restsumme, d. h. für einen Knoten $v_{i,j}$ ist

$$j = \beta - \sum_{\substack{k \geq i+1 \\ s_k \in S'}} s_k$$

Der Index j gibt also an, welche Summe die im Folgenden auszuwählenden Elemente noch haben müssen. Zu einem Element $v_{i,j} \in V_i$ unterscheiden wir nun zwei Fälle: Entweder s_i wird nicht in die Menge S' aufgenommen, dann ändert sich beim Übergang

von Schicht V_i nach V_{i-1} die Summe j nicht, es gibt also eine Kante zwischen $v_{i-1,j}$ und $v_{i,j}$. Alternativ wird s_i in S' aufgenommen, dann verringert sich beim Übergang nach V_{i-1} die Summe um s_i, das kennzeichnen wir durch eine Kante zwischen $v_{i-1,j-s_i}$ und $v_{i,j}$. Wir beginnen die Rekursion mit einem Knoten $t = v_{n,\beta}$ und versuchen einen Weg von t nach $s = v_{0,0}$ durch den Schichtgraphen zu konstruieren, dieser entspricht dann einem geeigneten S'.

Um den Sonderfall $\beta = 0$ mitbehandeln zu können, müssen wir identifizieren ob bzw. wie viele Elemente wir jeweils für unsere Teilmenge wählen. Dazu setzen wir ψ so, dass für jedes benutzte Element aus S ein Wert von 1 auf den aktuellen Knoten addiert wird. Wir wählen dann jeweils ein maximales α für jeden neu erreichten Knoten des Graphen. Wir finden auf diesem Weg die Teilmenge mit den meisten Elementen, die sich auf β aufsummiert. Sollte es eine solche nichtleere Teilmenge geben, so ergibt sich $\alpha(s) > 0$.

In Pseudocode erhalten wir folgenden Algorithmus für SUBSET SUM:

- Input: $S = \{s_1, ..., s_n\}$, β
- Output: $\alpha(s)$
  - $\alpha(s) \leq 0 \Rightarrow$ No-Instance
  - $\alpha(s) > 0 \Rightarrow$ Yes-Instance
- Define: $V = V_0 \cup ... \cup V_n$, where $V_0 = \{s = v_{0,0}\}$, $V_n = \{t = v_{n,\beta}\}$ and $V_i = \emptyset$ otherwise
- Initialize: $\alpha(t) = 0$, $\alpha(s) = -\infty$.
- For $i \in n, ..., 1$ do
  - For all $v_{i,j} \in V_i$ do
    - If $v_{i-1,j} \notin V_{i-1}$, create $v_{i-1,j}$, $V_{i-1} \leftarrow V_{i-1} \cup \{v_{i-1,j}\}$ and set $\alpha(v_{i-1,j}) = -\infty$.
    - If $v_{i-1,j-s_i} \notin V_{i-1}$, create $v_{i-1,j-s_i}$, $V_{i-1} \leftarrow V_{i-1} \cup \{v_{i-1,j-s_i}\}$ and set $\alpha(v_{i-1,j-s_i}) = -\infty$.
    - $\alpha(v_{i-1,j}) = \max\{\alpha(v_{i-1,j}), \alpha(v_{i,j})\}$
    - $\alpha(v_{i-1,j-s_i}) = \max\{\alpha(v_{i-1,j-s_i}), \alpha(v_{i,j}) + 1\}$

c) In jeder Schicht unseres Graphen gibt es höchstens $Sum := \sum_{i=1}^{n} |s_i|$ Knoten: Sei $S_P \subset S$ die Teilmenge aller positiven s_i, $S_N \subset S$ die Teilmenge aller negativen s_i, dann reichen uns Indices $\geq \sum_{x \in S_N} x$ und $\leq \sum_{x \in S_P} x$. Wir führen die innere Schleife pro Schicht also höchstens Sum mal aus und erhalten damit eine Anzahl von Operationen in $O(n \cdot Sum)$. Sei $Max := |\max_{x \in S} x|$. Da $Sum \leq nMax$, ist unsere Laufzeit also in $O(n^2 Max)$, also pseudo-polynomiell. (Man beachte jedoch, dass Sum bzw. Max nicht polynomiell, sondern exponentiell, in der Länge des in Bitdarstellung kodierten Inputs ist.)

Aufgabe 10.2
Ein umweltbewußter Student möchte zwar weiterhin mit seinem Auto von seinem Wohnheim in München an die TU nach Garching fahren, dabei aber nicht zuviel Benzin verbrauchen. Die möglichen Verbindungen seien durch einen Digraphen $G = (V, E)$ und zwei Knoten s (Studentenwohnung) und t (TUM) gegeben. Die Fahrzeiten und den Spritverbrauch je Kante geben zwei Abbildungen $l : E \to \mathbb{R}_{\geq 0}$ und $d : E \to \mathbb{N}_0$ (Spritverbrauch fällt nur in ganzzahligen Einheiten an). Der Student möchte für seine Fahrten einen Gesamtverbrauch von $D \in \mathbb{N}$ (einfache Fahrt) nicht überschreiten.

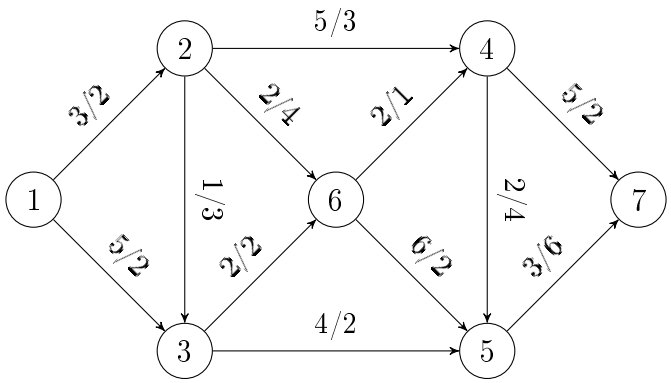
a) Bestimmen Sie in dem untenstehenden Graphen den kürzesten s-t-Weg ohne Beachtung des Verbrauchs. Welchen Spritverbrauch erfordert dieser Weg?

b) Sei $\xi_j(\delta)$ die Länge eines kürzesten s-j-Weges, auf dem höchstens δ Einheiten Sprit verbraucht werden. Formulieren Sie eine Rekursion für die $\xi_j(\delta)$.

c) Verwenden Sie die Rekursion aus b), um unter Verwendung von dynamischer Optimierung einen Algorithmus zu entwerfen, der für gegebenes D die Länge eines kürzesten s-t-Weges bestimmt, auf dem insgesamt nicht mehr als D Liter Sprit verbraucht werden. Skizzieren Sie zunächst einen geeigneten Schichtgraphen, definieren Sie Knoten und Kanten dieses Schichtgraphen und eine passende Stufenkostenfunktion.

d) Lösen Sie mit Ihrem Algorithmus das Problem auf untenstehendem Graphen für $D = 8$.

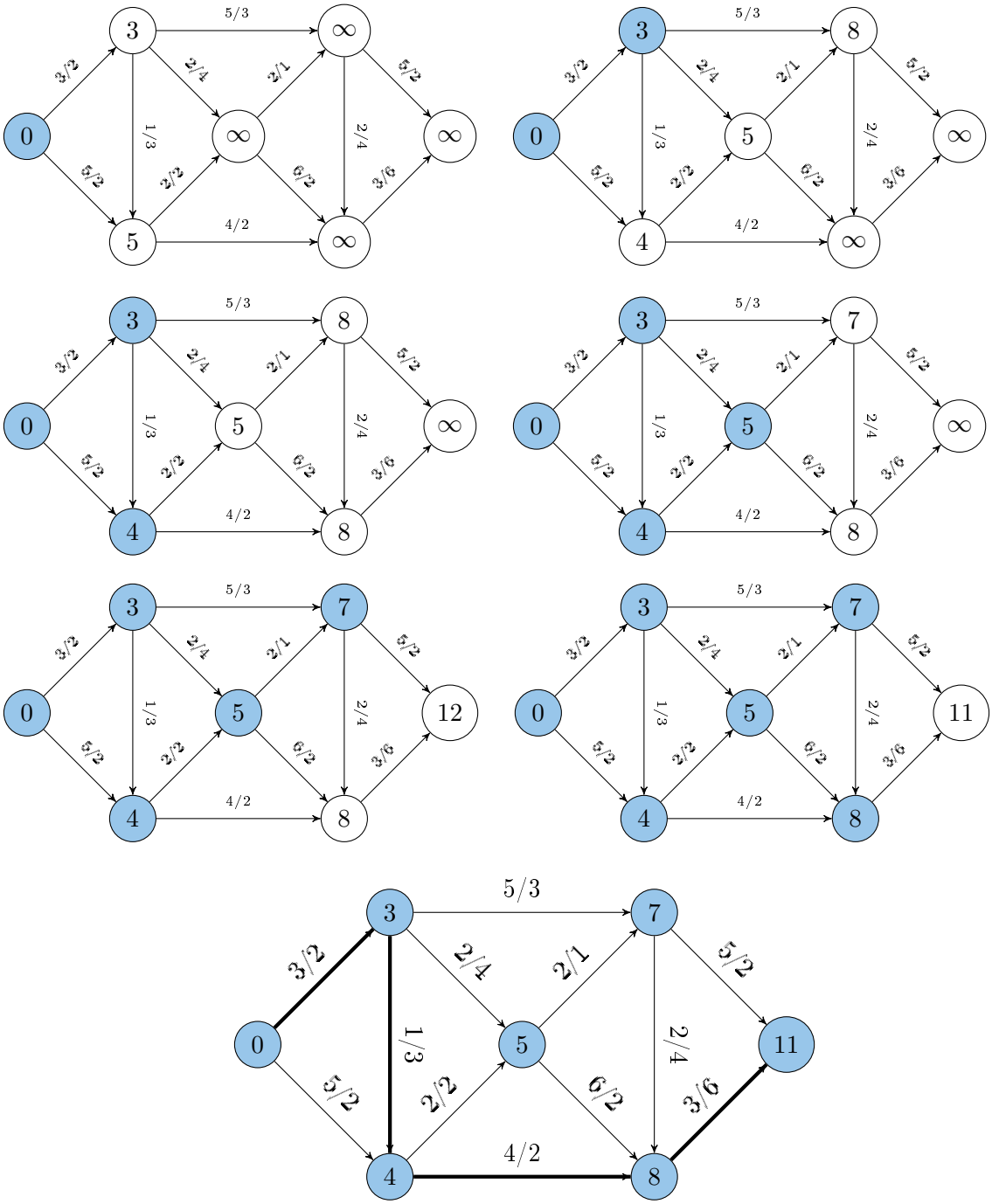
e) Wie hoch ist die Laufzeit Ihres Algorithmus? Bestimmen Sie eine möglichst gute obere Schranke in Abhängigkeit von den Größen im Input.



Beschriftung der Kanten: l_{ij}/d_{ij}

Lösung zu Aufgabe 10.2

a) Wir benutzen einen Dijkstra-Algorithmus zur Lösung des Problems, vgl. folgende graphische Darstellung. Als Beschriftung der Knoten verwenden wir hier das jeweilige Knotenlabel $\xi(v_i)$, blaue Knoten sind bereits endgültig gelabelt.



Der kürzeste Weg hat also Länge 11, der Spritverbrauch entlang des kürzesten Weges beträgt $2 + 3 + 2 + 6 = 13$.

b) Bei einem „Gesamtbudget“ von D bricht die Rekursion immer ab, wenn das Budget überschritten wird, also setzen wir $\xi_j(\delta) = \infty$ für alle $\delta > D$. Als Beginn der Rekursion eignet sich $\xi_s(0) = 0$. Für die Rekursion verwenden wir

$$\xi_j(\delta) := \min \left\{ \xi_j(\delta - 1), \min_{k \in N(j)} [\xi_k(\delta - \delta_{kj}) + l_{kj}] \right\}$$

Der Schichtgraph kann dann wie folgt definiert werden: In Schicht 0 befindet sich der Knoten $v_{0,s,0}$ (wir indizieren immer mit „Schicht,Knoten,gesamter Spritverbrauch“, $v_{0,s,0}$ entspricht also Knoten s in Schicht V_0 bei einem bisherigen Gesamtverbrauch von 0). Wenn Schicht V_{k-1} bekannt ist und einen Knoten $v_{k-1,i,\delta}$ enthält, so nehmen wir in Schicht V_k alle Knoten $v_{k,j,\delta'}$ auf mit $j \in N(i)$, für die $\delta' \geq \delta + d_{ij}$ gilt. Der Schichtgraph entspricht also im Wesentlichen dem fürs Kürzeste-Wege-Problem, wir nehmen aber zusätzlich eine Kopie jedes Knotens für jeden möglichen Gesamtverbrauch in die Schichten mit auf. Eine Kante kennzeichnet jeweils alle möglichen Übergänge von einer Schicht in die nächste, d. h. eine Kante $(v_{k-1,i,\delta}, v_{k,j,\delta'})$ existiert genau dann, wenn $(i,j) \in E$ und $\delta' \geq \delta + d_{ij}$. Die Stufenkostenfunktion ψ_k definieren wir als $\psi(v_{k-1,i,\delta}, (v_{k-1,i,\delta}, v_{k,j,\delta'})) := l_{ij}$. In der letzten Schicht k^* benötigen wir dann Knoten für den Originalknoten $t \in V$ und für jeden zulässigen Gesamtverbrauch, also z.B. alle Knoten $v_{k^*,t,\delta}$ mit $0 \leq \delta \leq D$.

c) Der Algorithmus, der sich aus dem oben skizzierten dynamischen Programm ergibt, kann als Abwandlung des Dijkstra-Algorithmus verstanden werden. Im Unterschied zu diesem benötigen wir jetzt nicht mehr ein Label (das beim Dijkstra-Algorithmus die Länge eines kürzesten s-j-Weges für jeden Knoten $j \in V$ angibt), sondern wir benutzen ein Label für jeden möglichen Spritverbrauch (das entspricht dem dritten Index der Knoten im Schichtgraphen). Wir benutzen also eine Abbildung $x : V \to \mathbb{R}^{D+1}$, $x_v = (\xi_v(0), \xi_v(1), \ldots, \xi_v(D))^T$. Das Update der Label im Dijkstra-Algorithmus führen wir unter Berücksichtigung des jeweiligen Kantenverbrauchs durch. Es ergibt sich damit Algorithmus 1.

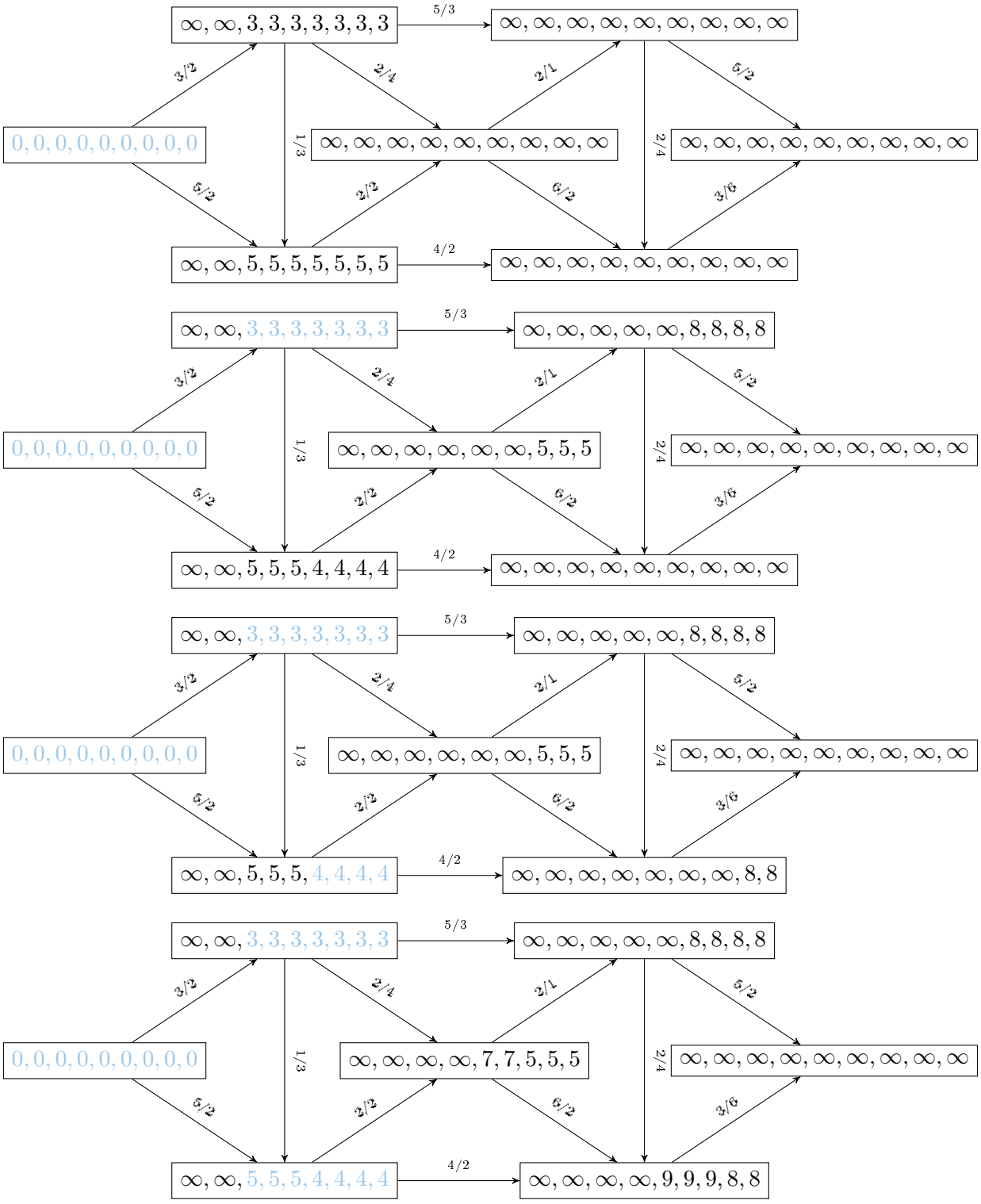
Algorithm 1: Dijkstra-Variante für Constrained Shortest Path

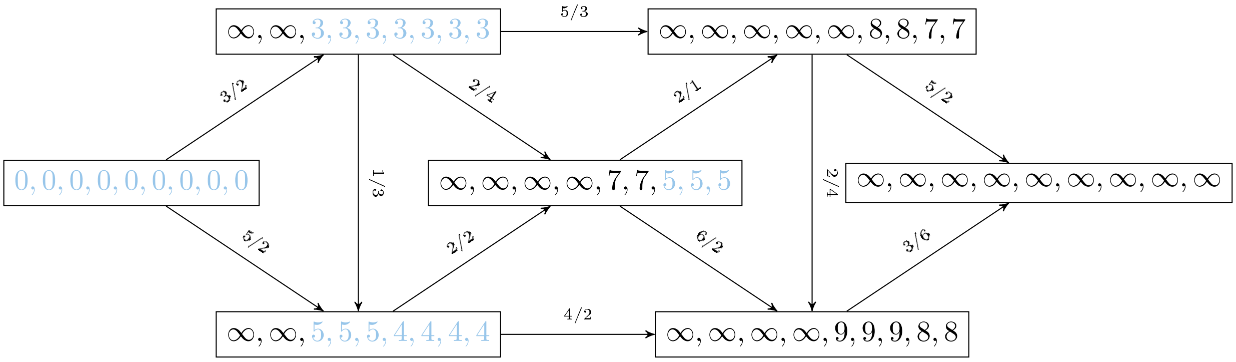
Input: Digraph $G = (V, E)$ mit Kantenbewertungen $l, d : E \to \mathbb{R}_{\geq 0}$, eine natürliche Zahl $D \in \mathbb{N}_0$ und $s \in V$.

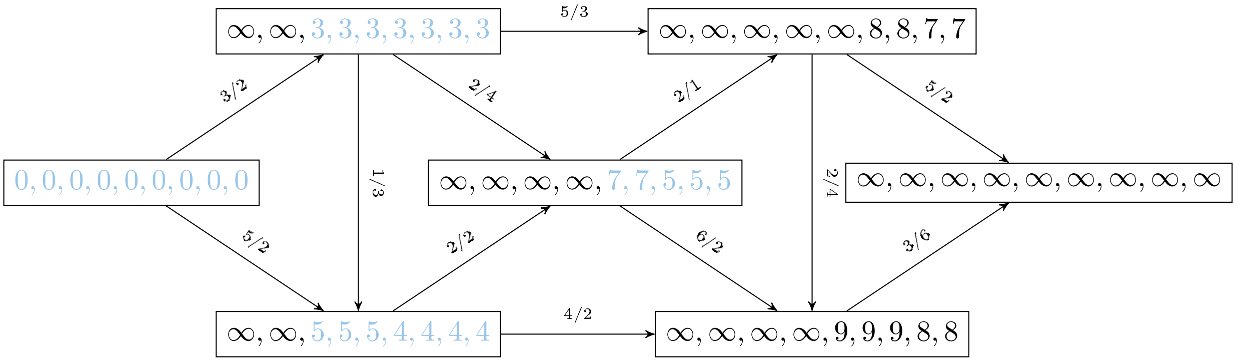
Output: Die Längen $x_v = (\xi_v(0), \xi_v(1), \ldots, \xi_v(D))^T$ eines kürzesten s-v-Weges für jeden Verbrauch in $0, 1, \ldots, D$ und jeden Knoten $v \in V$.

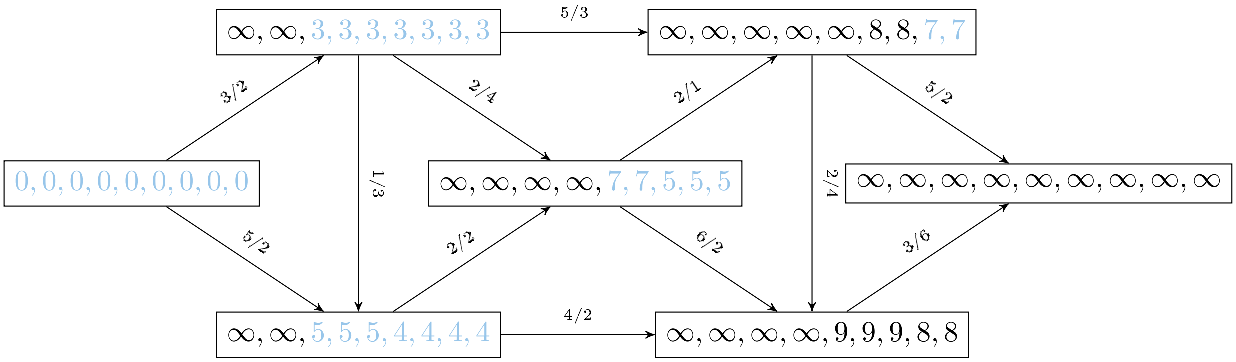
$S \leftarrow \{(s,0), \ldots, (s,D)\}$, $x_s \leftarrow 0 \cdot \mathbb{1}$, $x_v \leftarrow \infty \cdot \mathbb{1}$ für alle $v \in V \setminus \{s\}$
for $v \in N(G, s)$ **do**
  for $\delta = d_{sv}, \ldots, D$ **do**
    $\xi_v(\delta) \leftarrow l_{sv}$
  end
end
while *es gibt* $(v, \delta) \in V \times \{0, \ldots, D\} \setminus S$ *mit* $\xi_v(\delta) \neq \infty$ **do**
  Wähle $v^* \in V$, $\delta^* \in \{0, \ldots, D\}$, so dass
  $\xi_{v^*}(\delta^*) = \min\{\xi_v(\delta) : (v, \delta) \in V \times \{0, \ldots, D\} \setminus S\}$ und so, dass es kein $\delta' < \delta^*$ gibt, für das $\xi_{v^*}(\delta') = \xi_{v^*}(\delta^*)$.
  for $v \in N(G, v^*)$ **do**
    for $\delta \in \{\delta^* + d_{v^*,v}, \ldots, D\}$ **do**
      $\xi_v(\delta) \leftarrow \min\{\xi_v(\delta), \xi_{v^*}(\delta^*) + l_{v^*,v}\}$
    end
  end
  $S \leftarrow S \cup \{(v^*, \delta^*)\}$
end

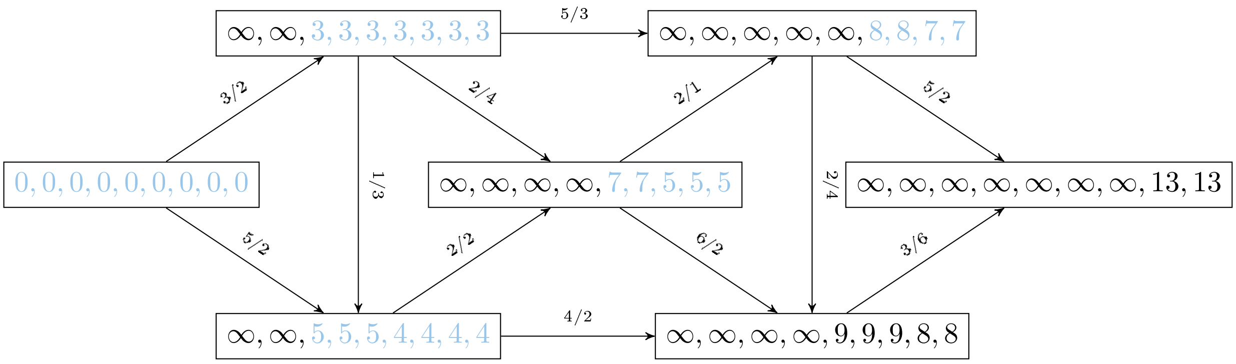
d) Wir stellen die Lösung wieder graphisch dar, endgültige Label sind blau gezeichnet. Die Label stehen jeweils in den Knoten und sind in der Form $\xi_v(0), \ldots, \xi_v(8)$ zu lesen. Außerdem fassen wir mehrere gleiche Schritte zu einem einzigen zusammen. Beachte: Es gibt mehrere „reine Labelling-Schritte“, bei denen keine tatsächliche Verbesserung mehr stattfindet. Das kann im Dijkstra-Algorithmus natürlich auch passieren, hier fällt es aber stärker auf.

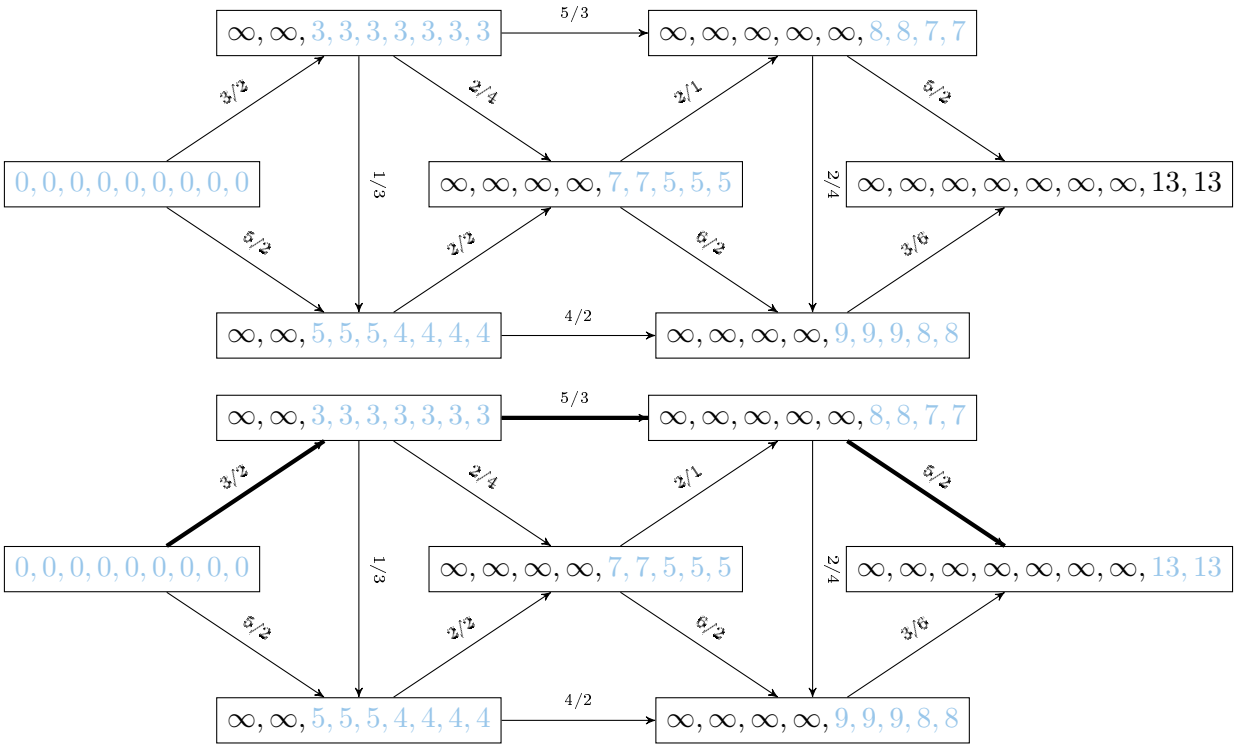



∞, ∞, 3, 3, 3, 3, 3, 3, 3
5/3
∞, ∞, ∞, ∞, ∞, 8, 8, 7, 7
3/2
2/4
2/1
5/2
0, 0, 0, 0, 0, 0, 0, 0, 0
1/3
∞, ∞, ∞, ∞, 7, 7, 5, 5, 5
2/4
∞, ∞, ∞, ∞, ∞, ∞, ∞, ∞, ∞
5/2
2/2
6/2
3/6
∞, ∞, 5, 5, 5, 4, 4, 4, 4
4/2
∞, ∞, ∞, ∞, 9, 9, 9, 8, 8


∞, ∞, 3, 3, 3, 3, 3, 3, 3
5/3
∞, ∞, ∞, ∞, ∞, 8, 8, 7, 7
3/2
2/4
2/1
5/2
0, 0, 0, 0, 0, 0, 0, 0, 0
1/3
∞, ∞, ∞, ∞, 7, 7, 5, 5, 5
2/4
∞, ∞, ∞, ∞, ∞, ∞, ∞, ∞, ∞
5/2
2/2
6/2
3/6
∞, ∞, 5, 5, 5, 4, 4, 4, 4
4/2
∞, ∞, ∞, ∞, 9, 9, 9, 8, 8


∞, ∞, 3, 3, 3, 3, 3, 3, 3
5/3
∞, ∞, ∞, ∞, ∞, 8, 8, 7, 7
3/2
2/4
2/1
5/2
0, 0, 0, 0, 0, 0, 0, 0, 0
1/3
∞, ∞, ∞, ∞, 7, 7, 5, 5, 5
2/4
∞, ∞, ∞, ∞, ∞, ∞, ∞, ∞, ∞
5/2
2/2
6/2
3/6
∞, ∞, 5, 5, 5, 4, 4, 4, 4
4/2
∞, ∞, ∞, ∞, 9, 9, 9, 8, 8


∞, ∞, 3, 3, 3, 3, 3, 3, 3
5/3
∞, ∞, ∞, ∞, ∞, 8, 8, 7, 7
3/2
2/4
2/1
5/2
0, 0, 0, 0, 0, 0, 0, 0, 0
1/3
∞, ∞, ∞, ∞, 7, 7, 5, 5, 5
2/4
∞, ∞, ∞, ∞, ∞, ∞, ∞, 13, 13
5/2
2/2
6/2
3/6
∞, ∞, 5, 5, 5, 4, 4, 4, 4
4/2
∞, ∞, ∞, ∞, 9, 9, 9, 8, 8



Der kürzeste Weg bei $D = 8$ hat also Länge 13.

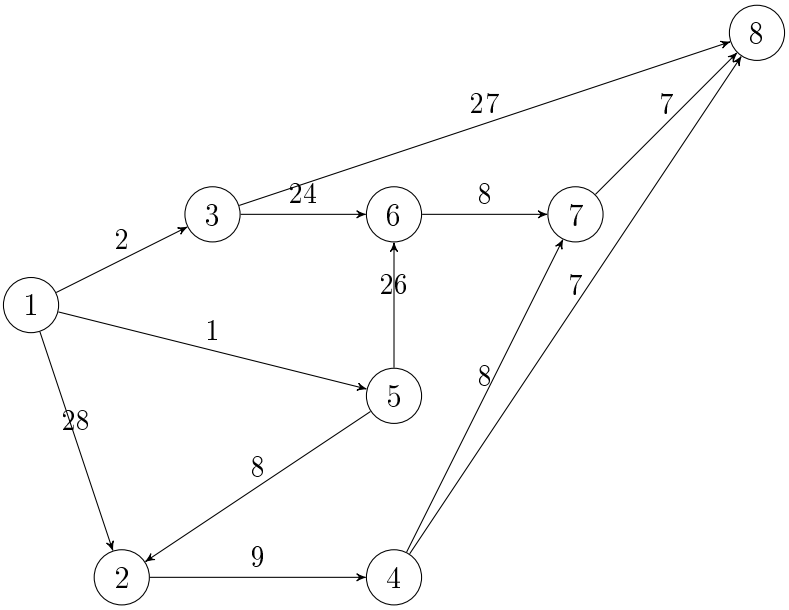
e) Die Laufzeit des Algorithmus liegt in $\mathcal{O}(n^2 D)$. Die Updates funktionieren im Grunde genauso wie im Dijkstra-Algorithmus, müssen aber für jedes der bis zu D Label durchgeführt werden, damit ist der Algorithmus nur noch pseudopolynomiell (da sich D mit Codierungslänge $\mathcal{O}(\log_2 D)$ codieren lässt). Wenn man den „Dijkstra-Teil“ geschickt implementiert (z. B. die Labels in einem entsprechend angepassten Heap verwaltet), lässt sich diese Schranke noch ein wenig verbessern (genauer gesagt, der n^2-Anteil), pseudopolynomiell bleibt sie aber trotzdem.

Aufgabe 10.3 *Hausaufgabe*
Gegeben sei der Graph $G = (V, E, c)$ mit $V = \{v_1, \ldots, v_8\}$, $E = \{e_1, \ldots, e_{12}\}$, wobei

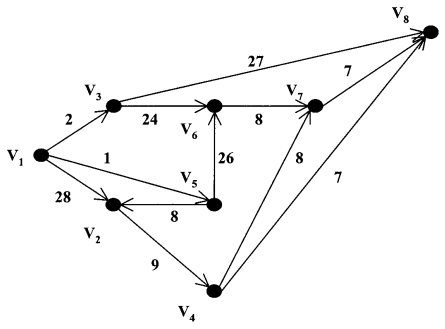
$$\begin{array}{lll} e_1 = (v_1, v_2), & e_2 = (v_1, v_3), & e_3 = (v_1, v_5), \\ e_4 = (v_2, v_4), & e_5 = (v_3, v_6), & e_6 = (v_3, v_8), \\ e_7 = (v_4, v_7), & e_8 = (v_4, v_8), & e_9 = (v_5, v_2), \\ e_{10} = (v_5, v_6), & e_{11} = (v_6, v_7), & e_{12} = (v_7, v_8), \end{array}$$

und $(c(e_1), \ldots, c(e_{12})) = (28, 2, 1, 9, 24, 27, 8, 7, 8, 26, 8, 7)$.

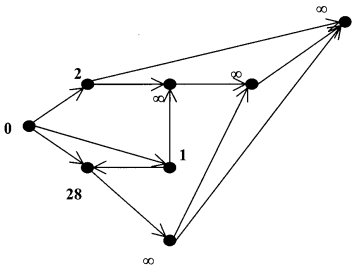
Berechnen Sie mit Hilfe des Dijkstra-Algorithmus den kürzesten Weg zwischen den Ecken v_1 und v_8. Skizzieren Sie jeden Zwischenschritt grafisch!

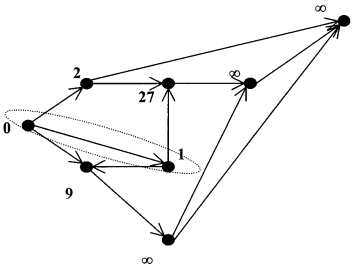


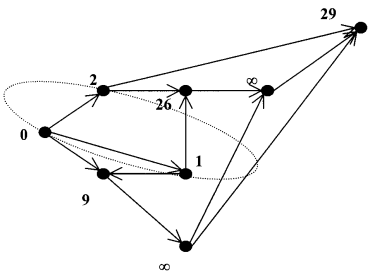
Lösung zu Aufgabe 10.3

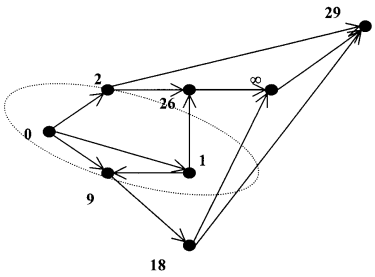


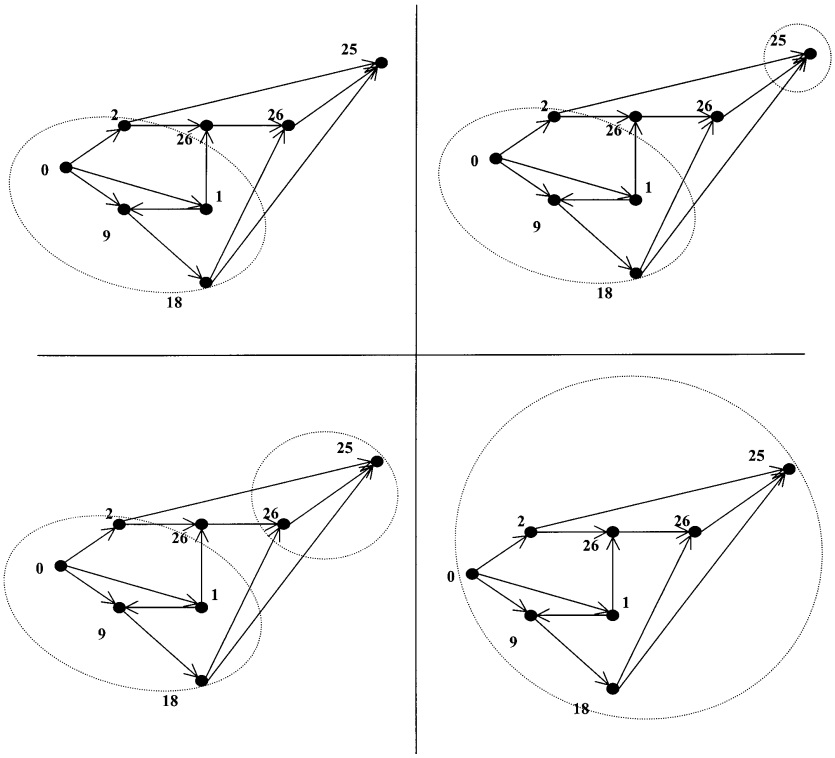
Initialisierung










25
2
26
26
0
1
9
18
25
2
26
26
0
1
9
18
25
2
26
26
0
1
9
18
25
2
26
26
0
1
9
18