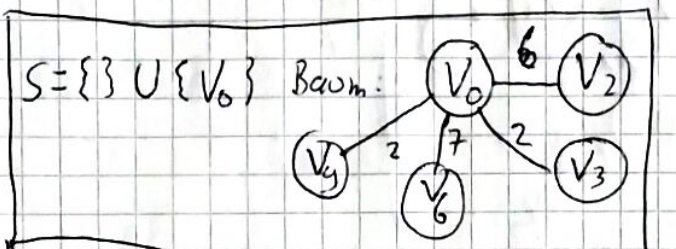
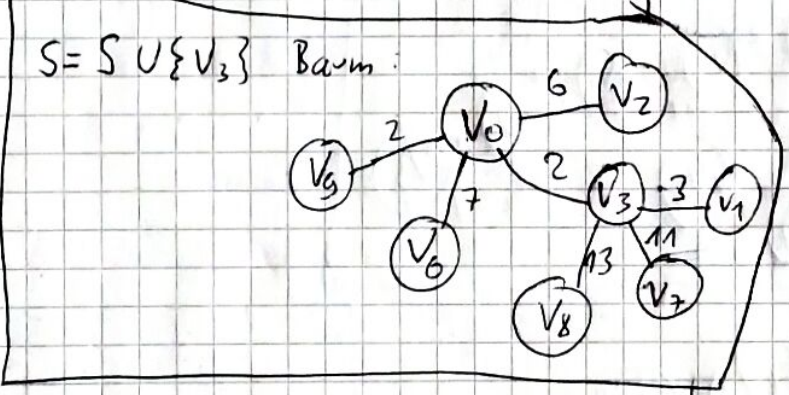
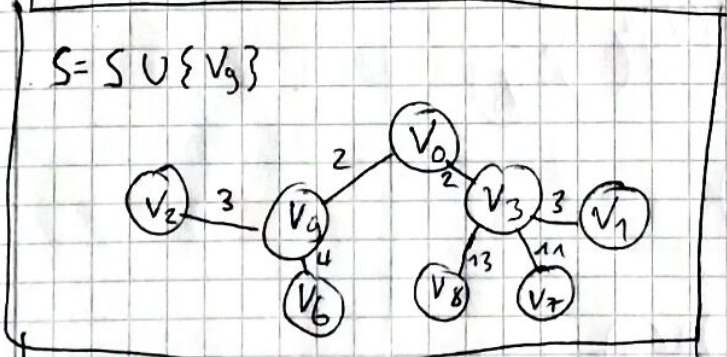
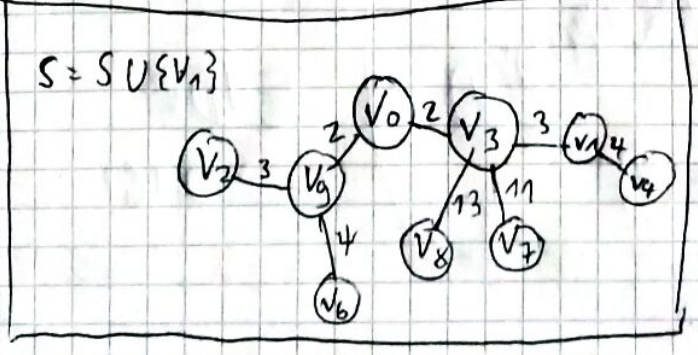
1a) 13/12

	V_0	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9
$S=\{V_0\}$	–	∞	6	2	∞	∞	7	∞	∞	2
$S=\{V_0, V_3\}$	–	5	6	–	∞	∞	7	11	13	2
$S=\{V_0, V_3, V_9\}$	–	5	5	–	∞	∞	6	11	13	–
$S=\{V_0, V_3, V_9, V_1\}$	–	–	5	–	9	∞	6	11	13	–
$S=\{V_0, V_3, V_9, V_1, V_2\}$	–	–	–	–	9	11	6	11	13	–
$S=\{V_0, V_3, V_9, V_1, V_2, V_6\}$	–	–	–	–	9	11	–	10	13	–
$S=\{V_0, V_3, V_9, V_1, V_2, V_4, V_6\}$	–	–	–	–	–	10	–	10	13	–
$S=\{V_0, V_3, V_9, V_1, V_2, V_4, V_5, V_6, V_7\}$	–	–	–	–	–	–	–	10	12	–
$S=\{V_0, V_3, V_9, V_1, V_2, V_4, V_5, V_6, V_7, V_8\}$	–	–	–	–	–	–	–	–	12	–

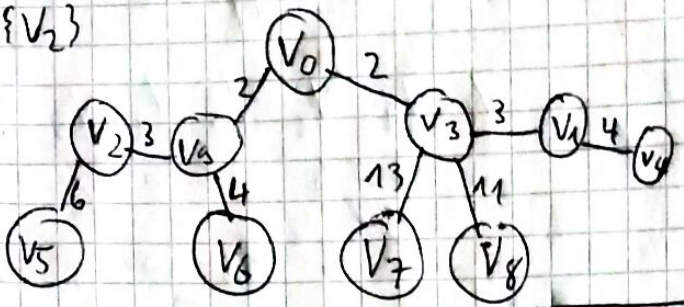




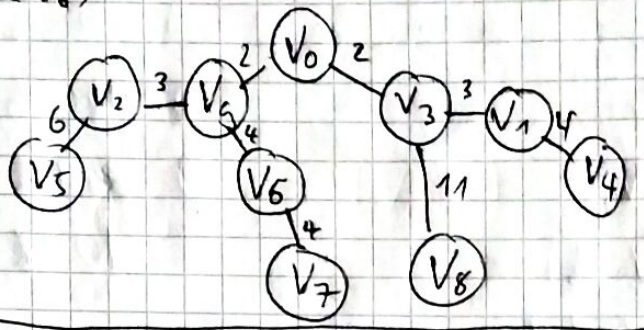




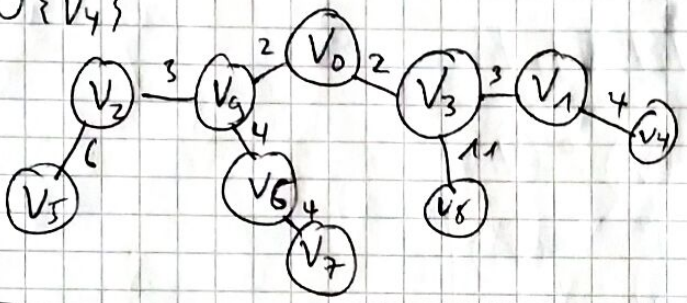
$S = S \cup \{V_2\}$



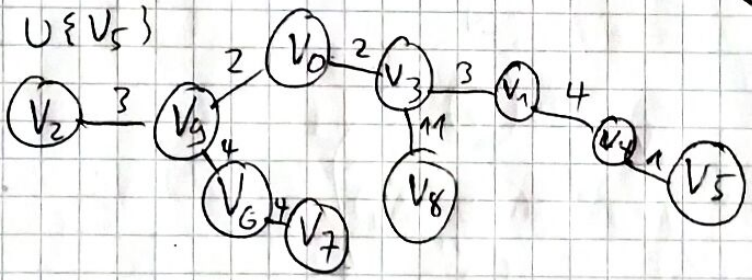
$S = S \cup \{V_6\}$



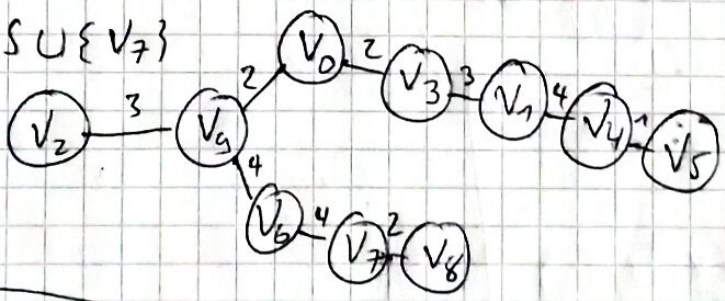
$S = S \cup \{V_4\}$



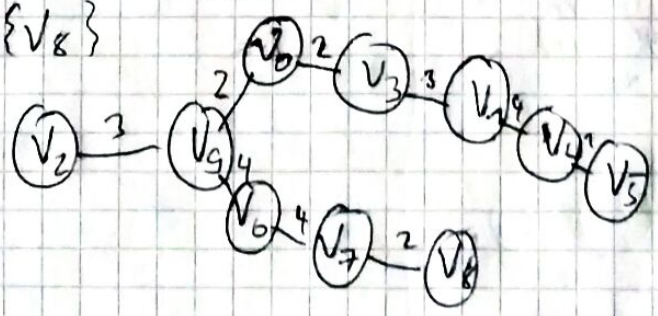
$S = S \cup \{V_5\}$



$S = S \cup \{V_7\}$



$S = S \cup \{V_8\}$

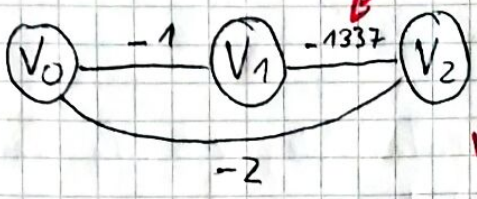


ENDE

→ OMG!

Voll! Beweis

1b)

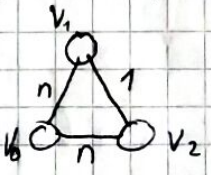


Mit Startknoten = V_0 ✓

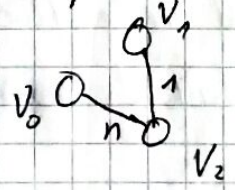
c) Sei v der Knoten, zu dem der kürzeste Weg gesucht ist.
Ersetze die "while" - Anweisung in (2) durch while (! v elem S), sodass der Algorithmus nur so lange läuft, wie der gesuchte Knoten nicht in S ist. Sobald v in S aufgenommen wird, steht der kürzeste Weg zu v fest. ✓

d) i) Da Dijkstra's Algorithmus einen Baum zurück gibt, der alle Knoten des Ausgangsgraphen enthält, und zusammenhängend ist, handelt es sich offensichtlicher Weise um einen ✓ Spannbaum.

ii) Es handelt sich nicht immer um einen minimalen Spannbaum.
Gegenbeispiel: Sei G =

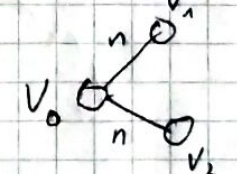


Dann ist ein minimaler Spannbaum von G:

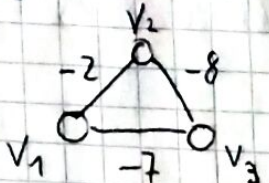


✓

ungleich dem von Dijkstra's Algorithmus erzeugten Baum:

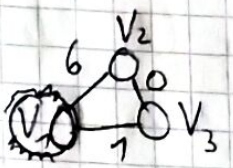


2a) Beispiel, das zeigt, dass das Verfahren nicht den kürzesten Weg findet:

G = 

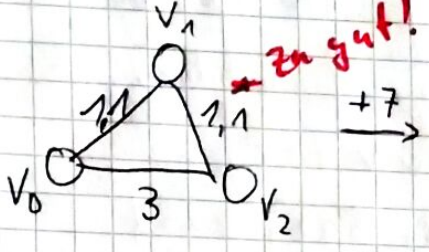
mit dem kürzesten Weg von V_1 nach V_3:
(V_1, V_2, V_3) Länge $= -10$

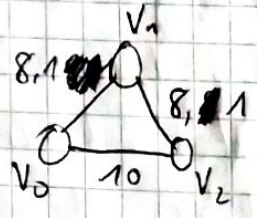
und G' mit $G + |-8|$ auf allen Kantengewichten:

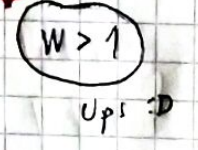


(V_1, V_3) Länge $= 1 \neq (V_1, V_2, V_3)$

b)

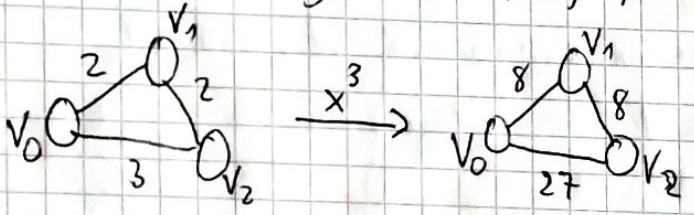






Kürzester Weg $(V_0, V_1, V_2) \neq (V_0, V_2)$

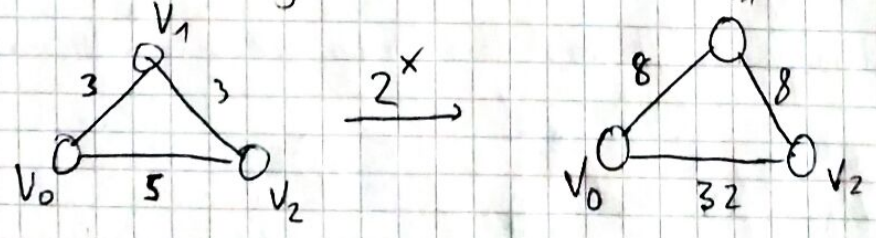
c)



$(V_0, V_2) \neq (V_0, V_1, V_2)$

d) Da alle Wege auf dem Graphen um den selben Faktor $c > 0$ wachsen, bleibt die Proportion zwischen der Länge der kürzesten Wege und alle anderen Wegen erhalten. Offensichtlicherweise bleiben dadurch immer noch die kürzesten Wege die kürzesten Wege.

e)



$(V_0, V_2) \neq (V_0, V_1, V_2)$

3a) Der Breitensuche-Algorithmus lässt die Kantengewichte außer Betracht. Er behandelt den Graphen so, als hätte jede Kante ein Gewicht von 1. ✓

b) Die einfachere Weise, die Breitensuche korrekt auf einem gewichteten Graphen auszuführen, wäre es, die Breitensuche auf einem vom Dijkstra Algorithmus erzeugten Baum auszuführen.
bsuche (dijkstra(v).root) $O(n \cdot \log(n))$, durch Dijkstra beschränkt.
Alternativ kann man die Kantengewichte durch "Pufferknoten" repräsentieren. Dies ist aber kompliziert zu implementieren, obwohl es sich Laufzeittechnisch ✓ lohnen würde $O(\text{sum(Kantengewichten)} \cdot |E| + \text{sum(Kantengewichten)} \cdot |V|)$.

Außerdem müssen alle Kantengewichte > 0 und $\in \mathbb{N}$ sein.

4) Zu sortierende Zahlen werden wie folgt aufgeteilt:
R1 erhält: 13, 44, 73, 4, 15, 22 → 4, 13, 15, 22, 44, 73
R2 erhält: 33, 99, 1, 17, 97, 24 → 1, 17, 24, 33, 97, 99
R3 erhält: 18, 35, 20, 10, 29, 8 → 8, 10, 18, 20, 29, 35
sortieren

Die Samples von R1 sind: 4, 15, 44
Die Samples von R2 sind: 1, 24, 97 und werden an R1 geschickt
Die Samples von R3 sind: 8, 18, 29 und werden an R1 geschickt
R1 sortiert die Samples: 1, 4, 8, 15, 18, 24, 29, 44, 97
Und sendet die Information [15, 29) an R2, dies ist das zu sortierende Intervall für R2.
R3 erhält das Intervall $[29, \infty)$
R1 sortiert alle Übrigen $(-\infty, 15)$

R1 sortiert also 13, 4, 1, 10, 8, R2 sortiert [illegible] 15, 22, 17, 24, 18, 20 und R3 sortiert 44, 73, 33, 99, 97, 35, 29.

Nachdem alle Rechner ihre Intervalle sortiert haben, werden die sortierten Intervalle von R2 und R3 an R1 geschickt, dieser setzt dann die erhaltenen Daten hinter sein sortiertes Intervall.

1, 4, 8, 10, 13, 15, 17, 18, 20, 22, 24, 29, 33, 35, 44, 73, 97, 99

R1 R2 R3