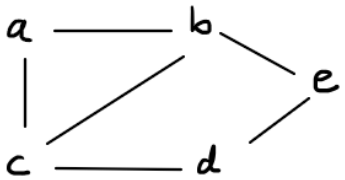
1. (3 Punkte) Gib für den abgebildeten Graphen die Implementierung in Python als Adjazenzmatrix an. Die Knoten a,b,c,.. sind auf die Indizes 0,1,2,... abgebildet.



Lösung:

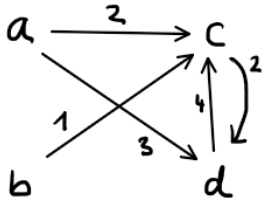
```
G = [[0,1,1,0,0],
     [1,0,1,0,1],
     [1,1,0,1,0],
     [0,0,1,0,1],
     [0,1,0,1,0]]
```

2. (3 Punkte) In der Variablen G sei ein ungewichteter Graph als Adjazenzmatrix implementiert. Die Knoten a,b,c... sind auf die Indizes 0,1,2... abgebildet. Schreibe eine Anweisung, die in der Variablen `nb` eine Liste mit allen Buchstaben speichert, die Nachbarn von b sind. Hinweis: die Funktion `chr(y+97)` ordnet den Zahlen y=0,1,2,... die Buchstaben a,b,c, ... zu.

Lösung:

```
nb = [chr(y+97) for y in range(len(G)) if G[1][y]]
```

3. (4 Punkte) Gib für den abgebildeten Graphen die Implementierung in Python als Adjazenzmatrix an. Die Knoten a,b,c,.. sind auf die Indizes 0,1,2,... abgebildet.



Lösung:

```
inf = float('inf')
G = [[0,inf,2,3],
     [inf,0,1,inf],
     [inf,inf,0,2],
     [inf,inf,4,0]]
```

4. (3 Punkte) In der Variablen G sei ein gewichteter Graph als Adjazenzmatrix implementiert. Die Knoten a,b,c... sind auf die Indizes 0,1,2... abgebildet. Schreibe eine Anweisung, die in der Variablen `nb` eine Liste mit allen Buchstaben speichert, die Nachbarn von a sind. Hinweis: die Funktion `chr(y+97)` ordnet den Zahlen y=0,1,2,... die Buchstaben a,b,c, ... zu.

Lösung:

```
nb = [chr(y+97) for y in range(len(G)) if y != 0 and G[0][y] < inf]
```

5. (6 Punkte) Der gewichtete Graph G ist mit einer Adjazenzmatrix implementiert. Die Knoten a,b,c,.. sind auf die Indizes 0,1,2,... abgebildet. Der Floyd-Algorithmus berechnet die abgebildeten Kosten- und Wegematrizen. Ermittle die Länge des kürzesten Weges von c nach f und gib die Länge der Teilstrecken an.

```
(Der Punkt steht für ∞)
Berechnete Kostenmatrix:
0   3   .   2   4   7
.   0   .   .   1   4
1   4   0   3   5   8
.   1   .   0   2   5
.   .   .   .   0   3
.   .   .   .   .   0
Berechnete Wegematrix:
0   3   0   0   1   4
1   1   1   1   1   4
2   3   2   0   1   4
3   3   3   3   1   4
4   4   4   4   4   4
5   5   5   5   5   5
```

Lösung:

```
Kürzester Weg von c nach f
(Kosten der Teilstrecken in Klammern)
c - a (1) - d (2) - b (1) - e (1) - f (3)
Gesamtkosten = 8
```

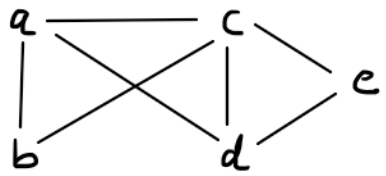
6. (6 Punkte) Berechne zu der Ausgangsmatrix D^{-1} des Floyd-Algorithmus die Matrix D^3

```
Ausgangsmatrix: (Der Punkt steht für ∞)
0   .   2   7   .
1   0   4   .   12
.   .   0   1   5
.   .   .   0   3
.   .   .   .   0
```

Lösung:

```
0   .   2   3   6
1   0   3   4   7
.   .   0   1   4
.   .   .   0   3
.   .   .   .   0
```

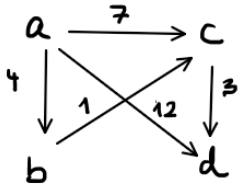
7. (3 Punkte) Gib für den abgebildeten Graphen die Implementierung in Python als ein dictionary von sets an.



Lösung:

```
G = {
 'a': set('bcd'),
 'b': set('ca'),
 'c': set('ade'),
 'd': set('ace'),
 'e': set('cd')
}
```

8. (4 Punkte) Gib für den abgebildeten Graphen die Implementierung in Python als ein dictionary von dictionaries an.



Lösung:

```
G = {
 'a': {'c':7, 'b':4, 'd':12},
 'b': {'c':1},
 'c': {'d':3},
 'd': {}
}
```

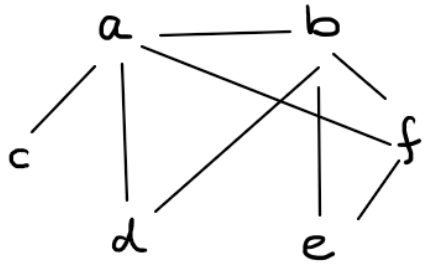
9. (6 Punkte) Der gerichtete, gewichtete Graph G sei als dictionary von dictionaries gegeben. Gehe mit Tiefensuche durch G und speichere im Verlauf der Tiefensuche in einen dictionary `ausgangskosten` für jeden Knoten die Summe der Kosten der von ihm wegführenden Kanten.

Lösung:

```
def explore(v):
    visited[v] = True
    for w in G[v]:
        ausgangskosten[v] += G[v][w]
    for w in G[v]:
        if not visited[w]:
            explore(w)

visited = {v : False for v in G}
ausgangskosten = {v: 0 for v in G}
for w in G:
    if not visited[w]:
        explore(w)
```

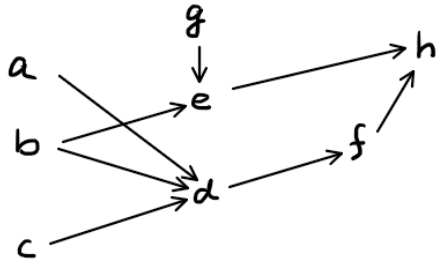
10. (3 Punkte) Der Graph wird von Startknoten a aus mit Tiefensuche traversiert. Die Nachbarn werden in alphabetischer Reihenfolge besucht. Schreibe an die Knoten ihre previsit/postvisit Nummern.



Lösung:

a	1	12
b	2	9
d	3	4
e	5	8
f	6	7
c	10	11

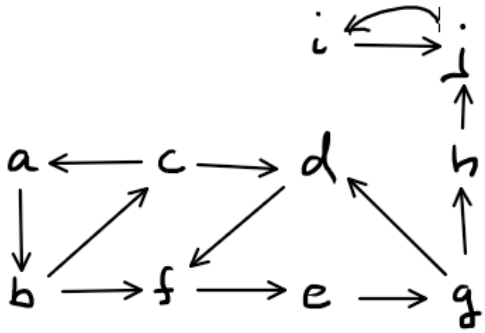
11. (4 Punkte) Der Graph wird mit dem im Unterricht vorgestellten Algorithmus topologisch sortiert. Welche Reihenfolge ergibt sich?



Lösung:

g c b e a d f h

12. (6 Punkte) Es sollen die starken Zusammenhangskomponenten des Graphen ermittelt werden. Der im Unterricht vorgestellte Algorithmus ermittelt dazu eine geeignete Reihenfolge, in der die Knoten des Graphen besucht werden. Die jeweils erreichbaren Knoten bilden dann eine starke Zusammenhangskomponente.



a. Gib diese Reihenfolge der Knoten an

Lösung:

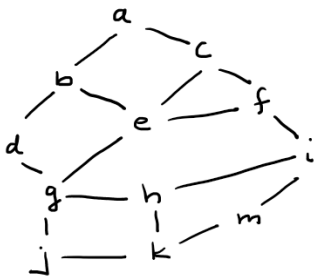
i j h d g e f a c b

b. Gib die Zusammenhangskomponenten in der Reihenfolge an, wie sie von dem im Unterricht vorgestellten Algorithmus entdeckt werden. Die Knotenreihenfolge innerhalb einer Komponente soll alphabetisch geordnet sein.

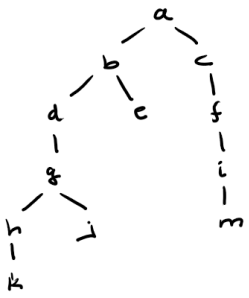
Lösung:

1 – i j
2 – h
3 – d e f g
4 – a b c

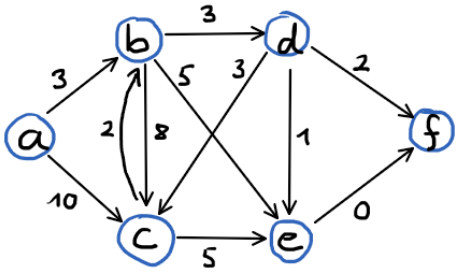
13. (3 Punkte) Zeichne den shortest-path Baum der Breitensuche. Wir beginnen bei a und nehmen an, dass Knoten auf der gleichen Ebene in alphabetischer Reihenfolge besucht werden.



Lösung:



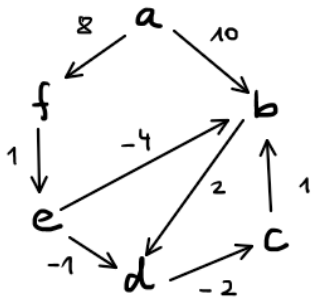
14. (6 Punkte) Ermittle mit dem Algorithmus von Dijkstra die kürzesten Wege von Knoten a zu allen anderen Knoten. Notiere die Reihenfolge der endgültig markierten Knoten. Notiere für jeden Knoten die Reihenfolge der Werte, mit denen er markiert wird.



Lösung:

```
Reihenfolge der endgültigen Markierungen: a b d e f c
a : 0
b : inf 3
c : inf 10 9
d : inf 6
e : inf 8 7
f : inf 8 7
```

15. (4 Punkte) Führe den Algorithmus von Bellman-Ford mit Startknoten a aus. Notiere nach jedem Durchgang die Kosten an den Knoten

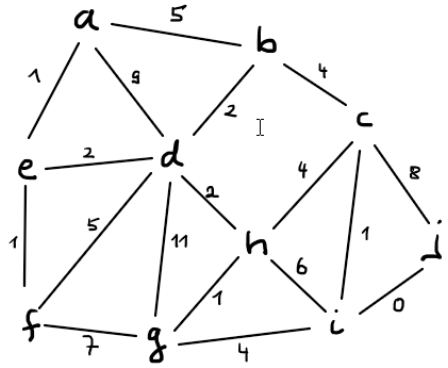


Lösung:

	a	b	c	d	e	f
0 :	0	inf	inf	inf	inf	inf
1 :	0	10	10	12	9	8
2 :	0	5	10	8	9	8
3 :	0	5	5	7	9	8
4 :	0	5	5	7	9	8

16. (6 Punkte) Bestimme für den Graphen einen minimalen Spannbaum mit dem Algorithmus von Kruskal. Immer wenn der Algorithmus uns eine Wahl lässt, wählen wir in alphabetischer Reihenfolge. Für eine Kante nennen wir die Knoten ebenfalls in alphabetischer Reihenfolge.

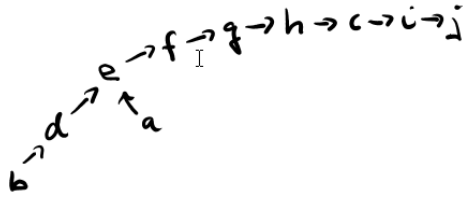
a. Gib die Kanten in der Reihenfolge an, in der sie gewählt werden.
b. Gib die Kosten des MST an.
c. Zeichne des Chefbaum des einfachen Algorithmus nach dem Verfahren aus dem Unterricht.
d. Zeichen den optimierten Chefbaum des Algorithmus mit path-compression und union by rank nach dem Verfahren aus dem Unterricht.



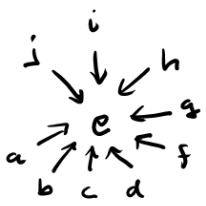
Lösung:

i–j, a–e, c–i, e–f, g–h, b–d, d–e, d–h, b–c
Gesamtkosten: 14

c.

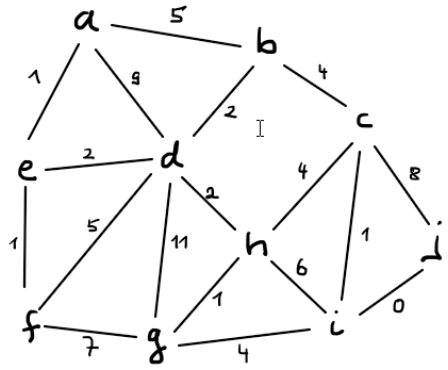


d.



17. (3 Punkte) Bestimme für den Graphen einen minimalen Spannbaum mit dem Algorithmus von Jarnik-Prim. Immer wenn der Algorithmus uns eine Wahl lässt, wählen wir in alphabetischer Reihenfolge. Das bedeutet u.a., dass wir von Knoten a aus starten.

a. Gib die Kanten in der Reihenfolge an, in der sie gewählt werden.
b. Gib die Kosten des MST an.



Lösung:

a–e, e–f, e–d, d–b, d–h, h–g, b–c, c–i, i–j
Gesamtkosten: 14