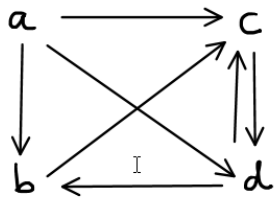
1. (4 Punkte) a. Gib für den abgebildeten Graphen die im Unterricht vorgeschlagenen Implementierung in Python an.
   b Mit welcher Schleife können wir alle Nachbarn des Knoten 'd' durchlaufen?
   c. Welcher boolesche Ausdruck überprüft, ob es eine Kante von 'c' nach 'a' gibt?
   Die Antworten für b. und c. sollen auch für andere Ausprägungen des Graphen gelten.



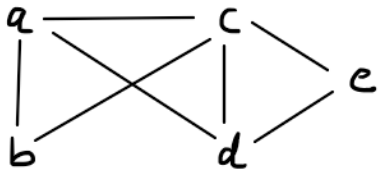
Lösung:

```
a. G = {
'a': set('bcd'),
'b': set('c'),
'c': set('d'),
'd': set('bc'),
}

b. for v in G['d']:
    # do something with v

c. 'a' in G['c']
```

2. (4 Punkte) a. Gib für den abgebildeten Graphen die im Unterricht vorgeschlagene Implementierung in Python an.
   b Mit welcher Schleife können wir alle Nachbarn des Knoten 'c' durchlaufen?
   c. Welcher boolesche Ausdruck überprüft, ob es eine Kante von 'a' nach 'd' gibt?
   Die Antworten für b. und c. sollen auch für andere Ausprägungen des Graphen gelten.



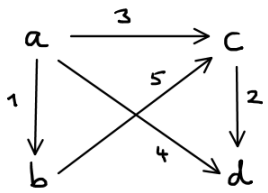
Lösung:

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

b. for v in G['c']:
     # do something with v

c. 'd' in G['a']
```

3. (5 Punkte) a. Gib für den abgebildeten Graphen die im Unterricht vorgeschlagene Implementierung in Python an.
   b Mit welcher Schleife können wir alle Nachbarn des Knoten 'a' durchlaufen?
   c. Welcher boolesche Ausdruck überprüft, ob es eine Kante von 'b' nach 'a' gibt?
   d. Durch welchen Ausdruck kommen wir an das Gewicht der Kante von 'b' nach 'c'?
   Die Antworten für b. c. und d. sollen auch für andere Ausprägungen des Graphen gelten.



Lösung:

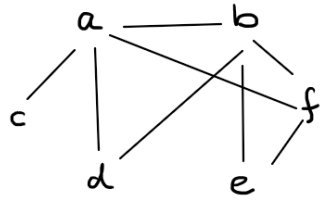
```
a.  G = {
        'a': {'b':1, 'c':3, 'd':4},
        'b': {'c':5},
        'c': {'d':2},
        'd': {}
    }

b.  for v in G['a']:
      # do something with v

c. 'a' in G['b']

d. G['b']['c']
```

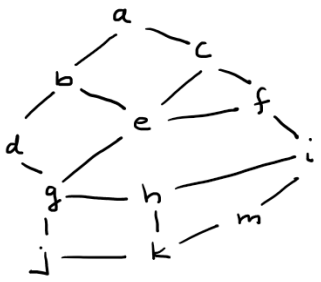
4. (2 Punkte) Der Graph wird von Startknoten a aus mit Tiefensuche traversiert. Die Nachbarn werden in alphabetischer Reihenfolge besucht. Gib die Reihenfolge der besuchten Knoten an.



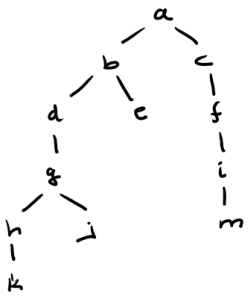
Lösung:

Reihenfolge: a b d e f c

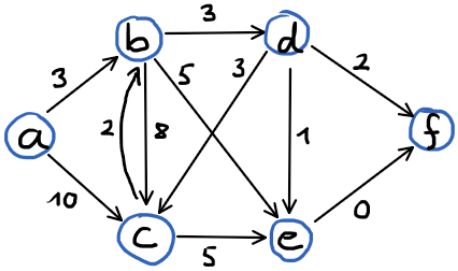
5. (3 Punkte) Zeichne den shortest-path Baum der Breitensuche. Wir beginnen bei a und nehmen an, dass Knoten auf der gleichen Ebene in alphabetischer Reihenfolge besucht werden.



Lösung:



6. (6 Punkte) Ermittle mit dem Algorithmus von Dijkstra die kürzesten Wege von Knoten a zu allen anderen Knoten.
   a. Notiere die Reihenfolge der endgültig markierten Knoten.
   b. 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
```

7. (6 Punkte) a. Bilde für das abgebildete grid den Graphen mit den Koordinaten als Knoten.

   b. Starte eine Breitensuche vom Knoten (3,3).

   Notiere für die Schleifendurchgänge 1 bis 5 den Inhalt von Q nach dem jedem Schleifendurchgang.

   Notiere nach dem Verfahren aus dem Unterricht im Graphen den Zustand der dictionaries dist und prev nach dem Schleifendurchgang 5. Wir gehen davon aus, dass die Nachbarn eines Knoten in der Reihenfolge oben, unten, links, rechts durchlaufen werden.

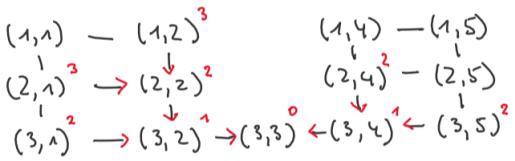
   c. Notiere Länge und Knotenabfolge des Pfads, den diese Breitensuche für einen Weg vom Startknoten nach (1,1) finden wird.

```
#######
#  #  #
#  #  #
#     #
#######
```

```
# Breitensuche:
dist[start] = 0
Q = deque([start])
while Q:
    u = Q.popleft()
    for v in G[u]:
        if dist[v] == inf:
            Q.append(v)
            dist[v] = dist[u]+1
            prev[v] = u
```

Lösung:

a.



b.

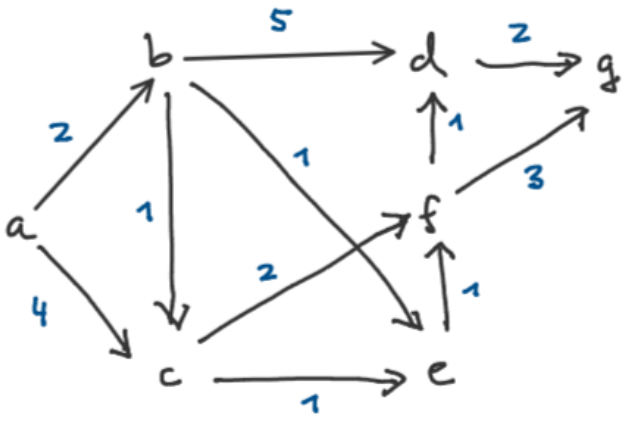
Q:	start	(3,3)
	1	(3,2) (3,4)
	2	(3,4) (2,2) (3,1)
	3	(2,2) (3,1) (2,4) (3,5)
	4	(3,1) (2,4) (3,5) (1,2) (2,1)
	5	(2,4) (3,5) (1,2) (2,1)

c. (3,3) (3,2) (2,2) (1,2) (1,1) Länge = 4

8. (6 Punkte) Starte den Algorithmus von Dijkstra vom Startknoten a.

   a. Notiere nach dem Verfahren aus dem Unterricht im Graphen den Zustand der dictionaries dist und prev nach Schleifendurchgang 5. Beachte, dass ein Schleifendurchgang auch über die continue-Anweisung beendet werden kann.

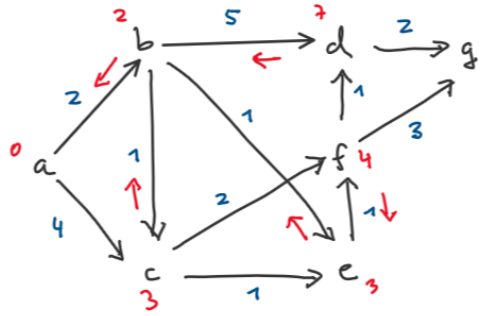
   b. Notiere für die Schleifendurchgänge 1 bis 5 den Inhalt des heaps vorlaeufig und des sets endgueltig. Beim heap reicht es, die Elemente in beliebiger Reihenfolge aufzuzählen.




```
# Die while-Schleife des Dijkstra-Algorithmus
while vorlaeufig:
    _, u = heappop(vorlaeufig)
    if u in endgueltig: continue
    endgueltig.add(u)
    for v in G[u]:
        if dist[v] > dist[u] + G[u][v]:
            dist[v] = dist[u] + G[u][v]
            prev[v] = u
            heappush(vorlaeufig,(dist[v],v))
```

Lösung:

a.



b. vorlaeufig:

endgueltig:

	vorlaeufig:	endgueltig:
start	(0,a) (∞,b-g)	
1	(2,b) (4,c) (∞,b-g)	a
2	(3,c) (3,e), (7,d) (4,c) (∞,b-g)	a, b
3	(3,e) (7,d) (4,c) (5,f) (∞,b-g)	a, b, c
4	(7,d) (4,c) (5,f) (4,f) (∞,b-g)	a, b, c, e
5	(7,d) (5,f) (4,f) (∞,b-g)	a, b, c, e