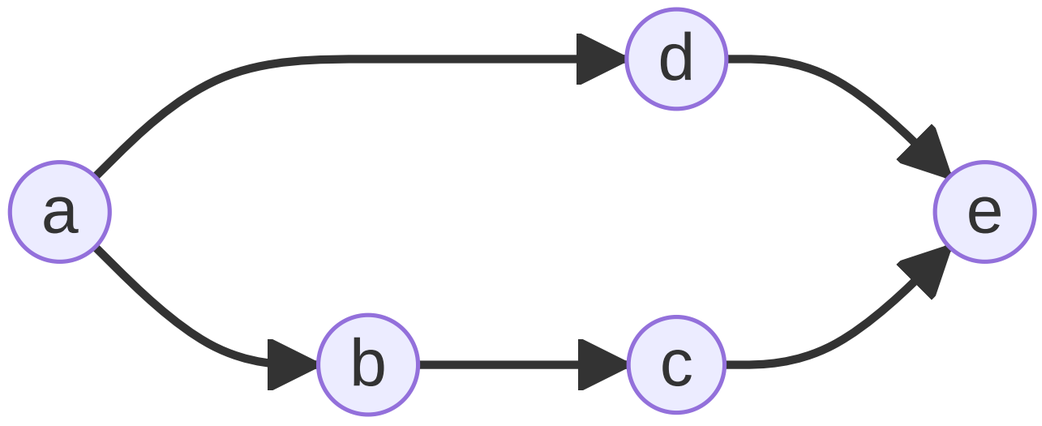
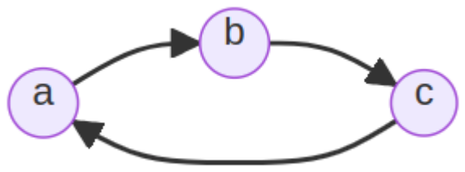
Topologisches Sortieren

Knoten eines gerichteten Graphen in eine Reihenfolge bring bei der alle Abhängigkeiten erfüllt werden.



Verboten sind zyklische Graphen



Der Algorithmus

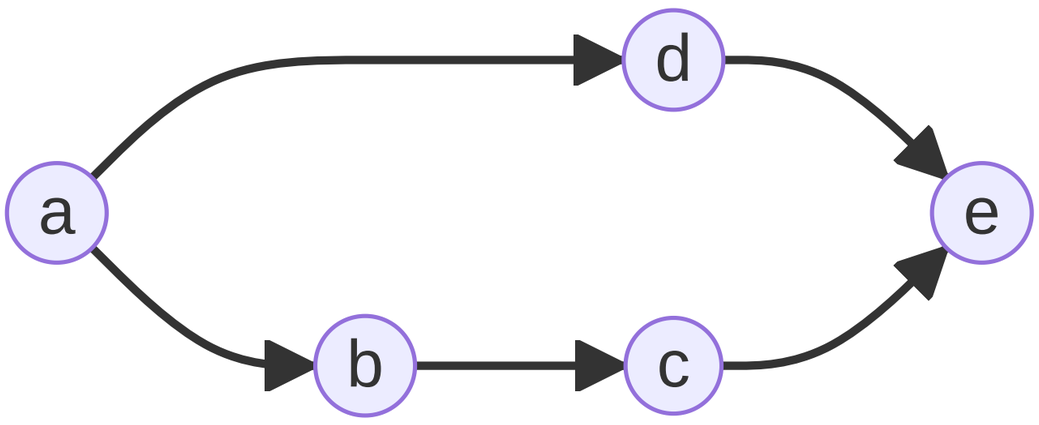
```
V: Menge von Knoten
E: Menge von Kanten (start, ende)
G: Gerichteter Graph

Funktion TopologischeSortierung(G = (V, E))
    L = leere Liste
    while V nicht leer do
        Zyklus = true
        for alle v in V do
            if v hat keine eingehenden Kanten then
                Zyklus = false
                Entferne v aus V
                Entferne alle Kanten die von v ausgehen aus E
                Füge v am Ende der Liste L hinzu
                print v
            endif
        endfor
        if Zyklus then
            print "Zyklus gefunden"
            break
        endif
    endwhile
end
```

In [ ]:
```python
def topologischeSortierung(G):
    V, E = G
    L = []
    while len(V) > 0:
        zyklus = True
        for v in V.copy():
            if len([e for e in E if e[1] == v]) == 0:
                zyklus = False
                V.remove(v)
                E = {e for e in E if e[0] != v}
                L.append(v)
        if zyklus:
            print("Zyklus gefunden")
            break
    return L
```

In [ ]:
```python
# Hier noch eine Debug-Version damit wir genauer sehen können was diese Funktion macht.
# Die Änderungen sind mit Kommentaren markiert.

from time import sleep as sleep
def topologischeSortierungDebug(G):
    callcounterg = 0 ##############################################################################################
    iterationcounter = 0 ##########################################################################################
    V, E = G
    L = []
    while len(V) > 0:
        callcounter = 0 ###########################################################################################
        iterationcounter += 1 #####################################################################################
        zyklus = True
        print(iterationcounter, end=". Durchgang, Aufgelöste Knoten: [ ") ########################################
        for v in V.copy():
            callcounter += 1 ######################################################################################
            if len([e for e in E if e[1] == v]) == 0:
                zyklus = False
                V.remove(v)
                E = {e for e in E if e[0] != v}
                L.append(v)
                print(v, end=" ") #################################################################################
                #sleep(0.5) #######################################################################################
        print("]   Knoten iteriert: ", callcounter, end=" ")  ####################################################
        callcounterg += callcounter ###############################################################################
        if zyklus:
            print("Zyklus gefunden")
            break
        print("         V: ", V, "E: ", E) ########################################################################
    print("Gesamtanzahl der iterierten Knoten: ", callcounterg, "Anzahl der Durchgänge: ", iterationcounter) #######
    return L
```



In [ ]:
```
V1 = ['a','b','c','d','e']
E1 = [
    ('a', 'b'),
    ('b', 'c'),
    ('a', 'd'),
    ('d', 'e'),
    ('c', 'e')
]
G1 = (V1.copy(), E1)
topologischeSortierungDebug(G1)
```

```
1. Durchgang, Aufgelöste Knoten: [ a b c d e ]   Knoten iteriert:  5          V:  [] E:  set()
Gesamtanzahl der iterierten Knoten:  5 Anzahl der Durchgänge:  1
```

Out[ ]:
```
['a', 'b', 'c', 'd', 'e']
```

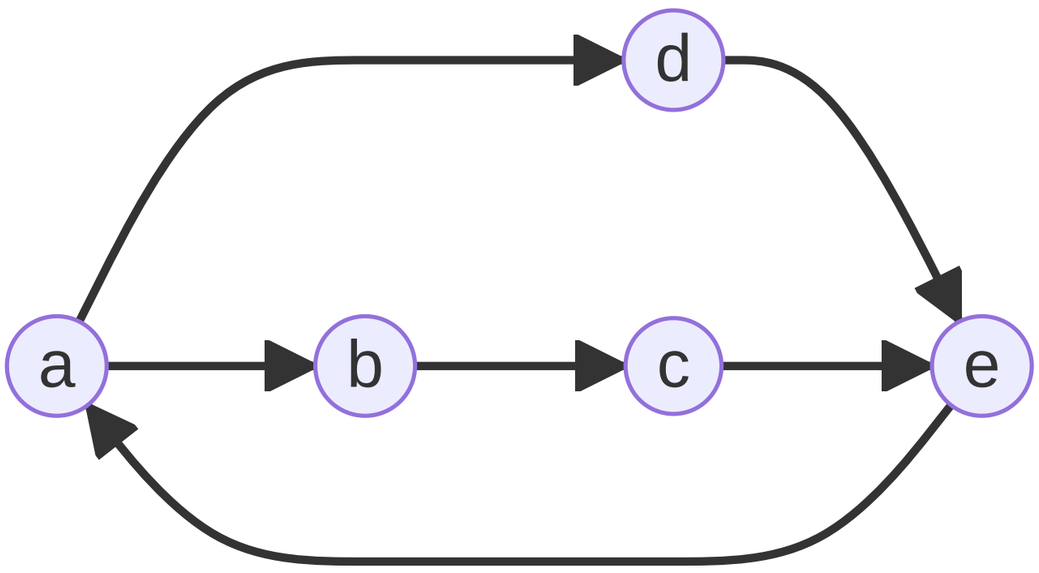
Achtung: Reihenfolge von V beeinflust Performance!

In [ ]:
```
V1reverse = ['d','e','c','b','a']
G1reverse = (V1reverse, E1)
topologischeSortierungDebug(G1reverse)
```

```
1. Durchgang, Aufgelöste Knoten: [ a ]   Knoten iteriert:  5          V:  ['d', 'e', 'c', 'b'] E:  {('b', 'c'), ('d', 'e'), ('c', 'e')}
2. Durchgang, Aufgelöste Knoten: [ d b ]   Knoten iteriert:  4          V:  ['e', 'c'] E:  {('c', 'e')}
3. Durchgang, Aufgelöste Knoten: [ c ]   Knoten iteriert:  2          V:  ['e'] E:  set()
4. Durchgang, Aufgelöste Knoten: [ e ]   Knoten iteriert:  1          V:  [] E:  set()
Gesamtanzahl der iterierten Knoten:  12 Anzahl der Durchgänge:  4
```

Out[ ]:
```
['a', 'd', 'b', 'c', 'e']
```

Hier ein Beispiel mit einem Zyklus

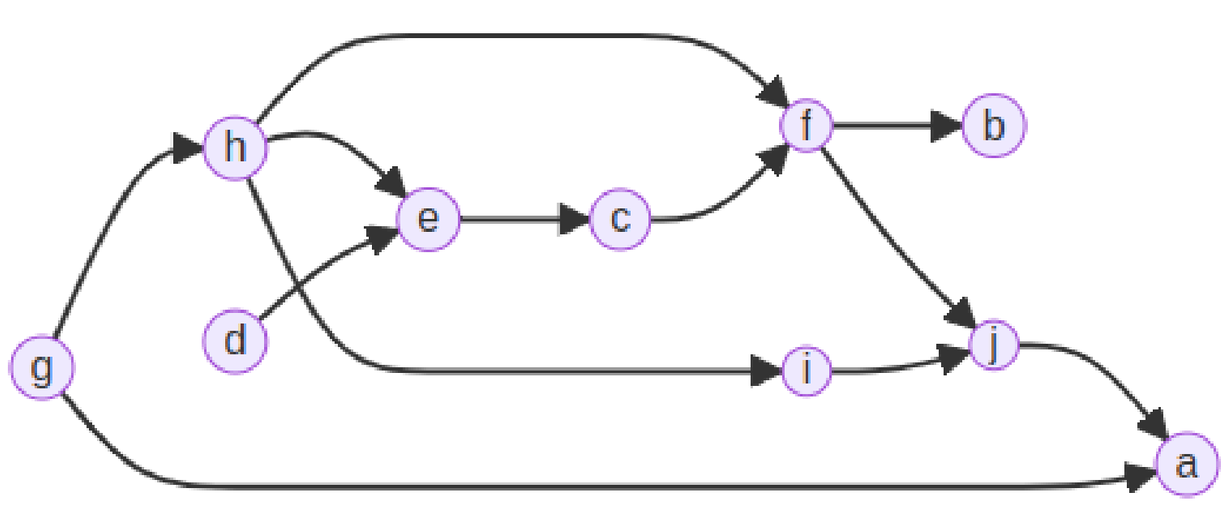


In [ ]:
```
E1z = [
    ('a', 'b'),
    ('b', 'c'),
    ('a', 'd'),
    ('d', 'e'),
    ('c', 'e'),
    ('e', 'a') # Zyklus
]
G1z = (V1, E1z)
topologischeSortierungDebug(G1z)
```

```
1. Durchgang, Aufgelöste Knoten: [ ]   Knoten iteriert:  5 Zyklus gefunden
Gesamtanzahl der iterierten Knoten:  5 Anzahl der Durchgänge:  1
```

Out[ ]:
```
[]
```

Jetzt noch ein komplexerer Graph

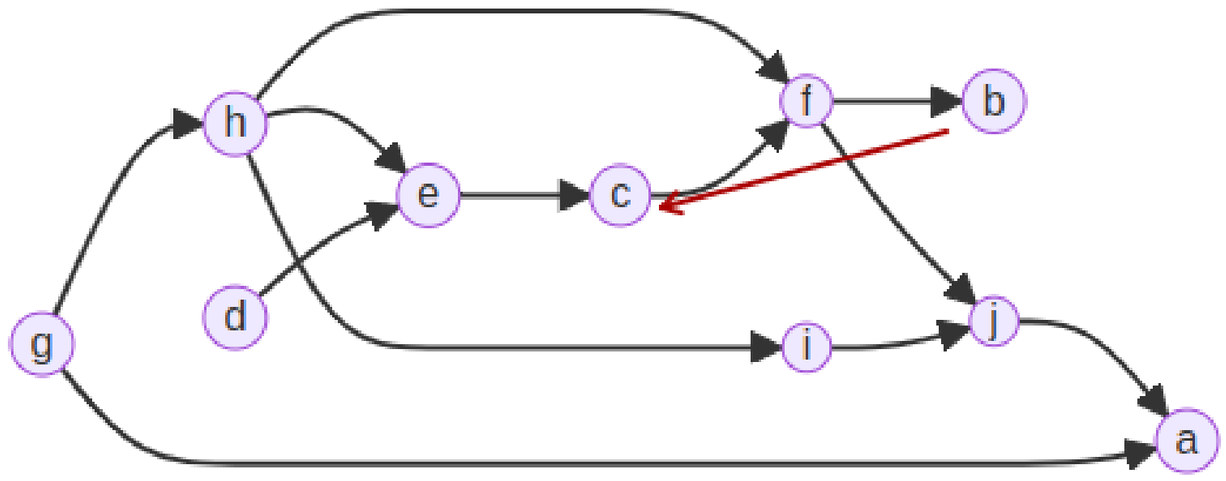


In [ ]:
```
V2 = ['a','b','c','d','e','f','g','h','i','j']
E2 = {('g', 'a'),('f', 'b'),('e', 'c'),('d', 'e'),('c', 'f'),('h', 'f'),('g', 'h'),('h', 'i'),('i', 'j'),('f', 'j'),('j', 'a'),('h', 'e')}
G2 = (V2.copy(), E2.copy())
topologischeSortierungDebug(G2)
```

```
1. Durchgang, Aufgelöste Knoten: [ d g h i ]   Knoten iteriert:  10          V:  ['a', 'b', 'c', 'e', 'f', 'j'] E:  {('e', 'c'), ('f', 'j'), ('f', 'b'), ('c', 'f'), ('j', 'a')}
2. Durchgang, Aufgelöste Knoten: [ e ]   Knoten iteriert:  6          V:  ['a', 'b', 'c', 'f', 'j'] E:  {('f', 'j'), ('f', 'b'), ('j', 'a'), ('c', 'f')}
3. Durchgang, Aufgelöste Knoten: [ c f j ]   Knoten iteriert:  5          V:  ['a', 'b'] E:  set()
4. Durchgang, Aufgelöste Knoten: [ a b ]   Knoten iteriert:  2          V:  [] E:  set()
Gesamtanzahl der iterierten Knoten:  23 Anzahl der Durchgänge:  4
```

Out[ ]: ['d', 'g', 'h', 'i', 'e', 'c', 'f', 'j', 'a', 'b']

Jetzt noch zyklisch



In [ ]:
```
E2z = {('g', 'a'),('f', 'b'),('e', 'c'),('d', 'e'),('c', 'f'),('h', 'f'),('g', 'h'),('h', 'i'),('i', 'j'),('f', 'j'),('j', 'a'),('h', 'e'),('b','c')}
G2z = (V2.copy(), E2z)
topologischeSortierungDebug(G2z)
```

```
1. Durchgang, Aufgelöste Knoten: [ d g h i ]   Knoten iteriert:  10          V:  ['a', 'b', 'c', 'e', 'f', 'j'] E:  {('e', 'c'), ('f', 'j'), ('f', 'b'), ('b', 'c'), ('c', 'f'),
('j', 'a')}
2. Durchgang, Aufgelöste Knoten: [ e ]   Knoten iteriert:  6          V:  ['a', 'b', 'c', 'f', 'j'] E:  {('f', 'j'), ('f', 'b'), ('b', 'c'), ('c', 'f'), ('j', 'a')}
3. Durchgang, Aufgelöste Knoten: [ ]   Knoten iteriert:  5 Zyklus gefunden
Gesamtanzahl der iterierten Knoten:  21 Anzahl der Durchgänge:  3
```

Out[ ]: ['d', 'g', 'h', 'i', 'e']

Wie nützen wir die Funktion mit unserem Graphenformat?

In [ ]:
```
Gdictset = {'a': set(), 'b': set(), 'c': {'f'}, 'd': {'e'}, 'e': {'c'}, 'f': {'b', 'j'}, 'g': {'a', 'h'}, 'h': {'e', 'f', 'i'}, 'i': {'j'}, 'j': {'a'}}

def dictset2tuple(G):
    V = list(G.keys())
    E = set()
    for k in G:
        for v in G[k]:
            E.add((k,v))
    return (V,E)

def tuple2dictset(G):
    V, E = G
    G = {v:set() for v in V}
    for e in E:
        G[e[0]].add(e[1])
    return G

print( tuple2dictset(dictset2tuple(Gdictset)) == Gdictset )
dictset2tuple(Gdictset)
```

```
True
```

Out[ ]:
```
(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j'],
 {('c', 'f'),
  ('d', 'e'),
  ('e', 'c'),
  ('f', 'b'),
  ('f', 'j'),
  ('g', 'a'),
  ('g', 'h'),
  ('h', 'e'),
  ('h', 'f'),
  ('h', 'i'),
  ('i', 'j'),
  ('j', 'a')})
```

In [ ]:
```
topologischeSortierungDebug(dictset2tuple(Gdictset))
```

```
1. Durchgang, Aufgelöste Knoten: [ d g h i ]   Knoten iteriert:  10          V:  ['a', 'b', 'c', 'e', 'f', 'j'] E:  {('e', 'c'), ('f', 'j'), ('f', 'b'), ('c', 'f'), ('j', 'a')}
2. Durchgang, Aufgelöste Knoten: [ e ]   Knoten iteriert:  6          V:  ['a', 'b', 'c', 'f', 'j'] E:  {('f', 'j'), ('f', 'b'), ('j', 'a'), ('c', 'f')}
3. Durchgang, Aufgelöste Knoten: [ c f j ]   Knoten iteriert:  5          V:  ['a', 'b'] E:  set()
4. Durchgang, Aufgelöste Knoten: [ a b ]   Knoten iteriert:  2          V:  [] E:  set()
Gesamtanzahl der iterierten Knoten:  23 Anzahl der Durchgänge:  4
```

Out[ ]: ['d', 'g', 'h', 'i', 'e', 'c', 'f', 'j', 'a', 'b']