Python-Code der Tiefensuche

```
def dfs(G, v0):
    '''Gibt das Ergebnis einer Tiefensuche mit
    Startknoten v0 auf dem Graphen G (Dictionary
    of Lists) zurück'''

    def recurse(G, v):
        B.append(v)
        for w in G[v]:
            if w not in B: # O(n) -> ineffizient
                recurse(G, w)

    B = [] # Liste der besuchten Knoten
    recurse(G, v0)
    return B
```

Python-Code der Breitensuche

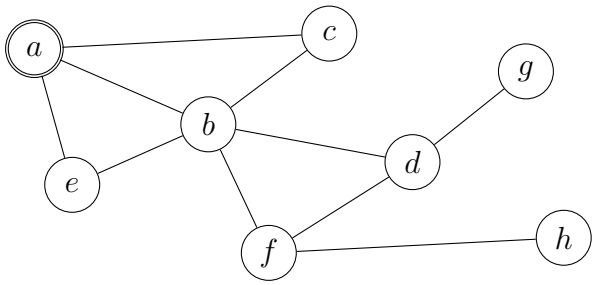
```
def bfs(G, v0):
    '''Gibt das Ergebnis einer Breitensuche
    mit Startknoten v0 auf dem Graphen G
    (Dictionary of Lists) zurück'''

    B = [v0] # Liste der besuchten Knoten
    Q = [v0] # naive Warteschlange

    while Q != []:
        v = Q.pop(0)     # ineffizient!
        for w in G[v]:
            if w not in B:
                B.append(w)
                Q.append(w)

    return B
```

Aufgabe 1



$\Rightarrow a, b, c, d, f, h, g, e$

Aufgabe 2

Bestimme den Aufwand kumulativ aus den Teilen des Algorithmus:

(a) Da der Graph zusammenhängend ist, wird jeder der $|V|$ Knoten genau einmal in die Liste der besuchten Knoten aufgenommen, was eine erneute Verarbeitung verhindert.

$\Rightarrow O(|V|)$

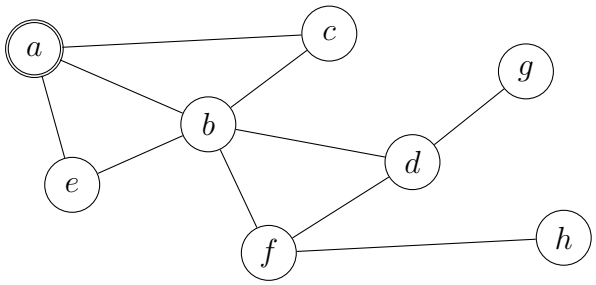
(b) Da von jedem Knoten v aus $\deg(v)$ Nachbarn getestet werden, ergibt dies wegen des Satzes über den Grad eines Graphen zusätzlich einen Aufwand proportional zu

$$\sum_{v \in V} \deg(v) = 2 \cdot |E|$$

$\Rightarrow O(|E|)$

Gesamtaufwand: $O(|V| + |E|)$

Aufgabe 3



Entferne den Knoten x aus der Warteschlange Q, füge ihn am Ende der Liste L der besuchten Knoten ein und füge alle seine noch nicht besuchten Nachbarn (sofern es solche gibt) *in alphabetischer Reihenfolge* in die Warteschlange Q ein.

(ein) Q (aus)		L	
a	$\rightarrow$	a	
e, c, b	$\rightarrow$	b	
f, d, e, c	$\rightarrow$	c	
f, d, e	$\rightarrow$	e	
f, d	$\rightarrow$	d	
g, f	$\rightarrow$	f	
h, g	$\rightarrow$	g	
h	$\rightarrow$	h	
–			Q ist leer $\rightarrow$ Ende

$L \Rightarrow a$, b, c, e, d, f, g, h

Aufgabe 4

Bestimme den Aufwand kumulativ aus den Teilen des Algorithmus.

(a) Da der Graph zusammenhängend ist, wird jeder Knoten, der sich noch nicht in der Liste L befindet, genau einmal

- der Liste L hinzugefügt
- ans Ende der Warteschlange Q gesetzt
- aus der Warteschlange Q entfernt

$\Rightarrow O(|V|)$

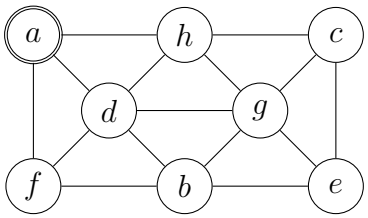
(b) Für jedem Knoten v müssen $\deg(v)$ Nachbarn getestet werden, ob sie bereits in L sind. Wegen des Satz über den Grad eines Graphen ist dieser Aufwand proportional zu

$$\sum_{v \in V} \deg(v) = 2 \cdot |E|$$

$\Rightarrow O(|E|)$

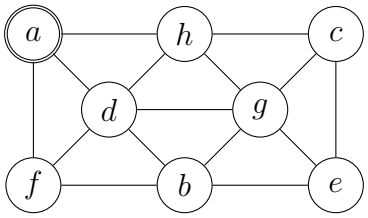
Gesamtaufwand: $O(|V| + |E|)$.

Aufgabe 5



$\Rightarrow a, d, b, e, c, g, h, f$

Aufgabe 6



Entferne den Knoten x aus der Warteschlange Q, füge ihn am Ende der Liste L der besuchten Knoten ein und füge alle seine noch nicht besuchten Nachbarn (sofern es solche gibt) *in alphabetischer Reihenfolge* in die Warteschlange Q ein.

(ein) Q (aus)		L	
a	$\rightarrow$	a	
h, f, d	$\rightarrow$	d	
g, b, h, f	$\rightarrow$	f	
g, b, h	$\rightarrow$	h	
c, g, b	$\rightarrow$	b	
e, c, g	$\rightarrow$	g	
e, c	$\rightarrow$	c	
e	$\rightarrow$	e	
–			Q ist leer $\rightarrow$ Ende

$L \Rightarrow a, d, f, h, b, g, c, e$