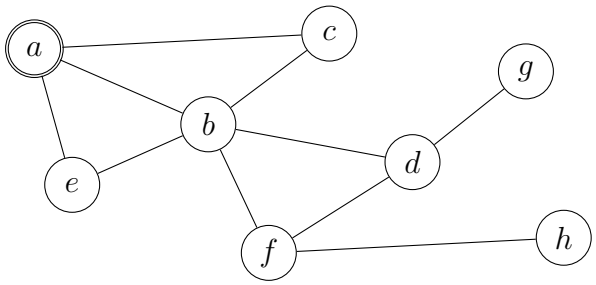
```
-----------------------------------------------
Algorithmus, der mittels Tiefensuche alle Knoten
eines Graphen G besucht, die von einem Knoten s
aus erreichbar sind.
-----------------------------------------------

Input:  Graph G, Startknoten s
Output: Liste der besuchten Knoten B

B = []  # Liste aller bereits besuchten Knoten
v = s   # aktueller Knoten
Führe die folgende Funktion f(G, v) aus:
    Setze v ans das Ende von B  # v wurde 'besucht'
    for w in Adj(v):  # Adj(v) sind sortiert
        wenn w nicht in B liegt:
            rufe f(G, w) rekursiv auf
Gib B aus

-----------------------------------------------
Algorithmus, der mittels Breitensuche alle Knoten
eines Graphen G besucht, die von einem Knoten s
aus erreichbar sind.
-----------------------------------------------

Input:  Graph G, Startknoten s
Output: Liste der besuchten Knoten B

B = [] # Liste der besuchten Knoten
Q = leere Warteschlange
B.append(s)
Q.enqueue(s)
while Q != leer:
    v = Q.dequeue()
    for w in Adj(v): # Adj(v) sind sortiert
        if w not in B:
            B.append(w)
            Q.enqueue(w)
Gib B aus
```

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. Dabei wird pro Knoten ein konstanter Aufwand c_1 betrieben, weshalb der Aufwand $c_1 \cdot |V|$ beträgt.

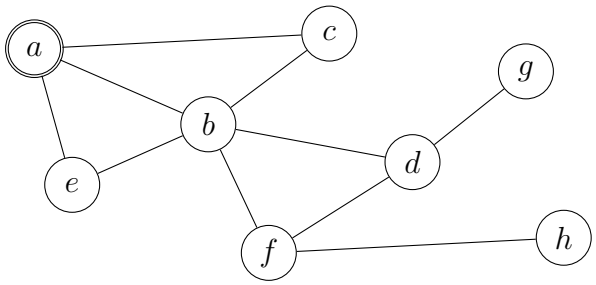
(b) Da von jedem Knoten v aus $\deg(v)$ Nachbarn getestet werden, ergibt dies zusätzlich einen Aufwand proportional zu

$$\sum_{v \in V} \deg(v) = 2 \cdot |E| \quad \text{(Satz über den Grad eines Graphen)}$$

was insgesamt zu einem Aufwand von $2 \cdot c_2 \cdot |E|$ führt, wobei die Konstante c_2 summarisch für den Aufwand steht, den nächsten Knoten in der Adjazenzliste aufzurufen und seinen Verarbeitungsstatus zu testen.

Gesamtaufwand: $O(c_1|V| + c_2|E|) = O(|V| + |E|)$.

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

Der Aufwand dafür beträgt $c_1 \cdot |V|$.

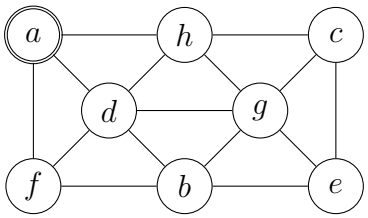
(b) Von jedem Knoten v müssen $\deg(v)$ Nachbarn getestet werden, ob sie bereits in L sind. Wegen

$$\sum_{v \in V} \deg(v) = 2 \cdot |E|, \quad \text{(Satz über den Grad eines Graphen)}$$

beträgt der Aufwand dafür $2 \cdot c_2 \cdot |E|$, wobei c_2 für den Aufwand steht, den nächsten Knoten in der Adjazenzliste aufzurufen und seinen Verarbeitungsstatus zu testen.

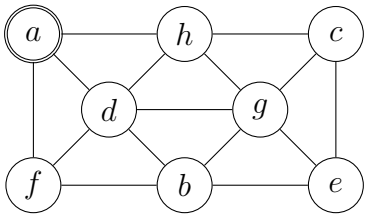
Gesamtaufwand: $O(c_1|V| + c_2|E|) = O(|V| + |E|)$.

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

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