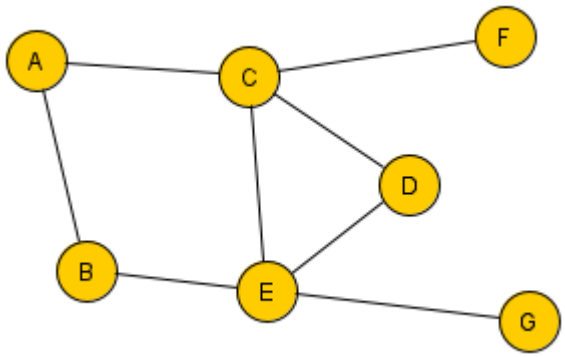
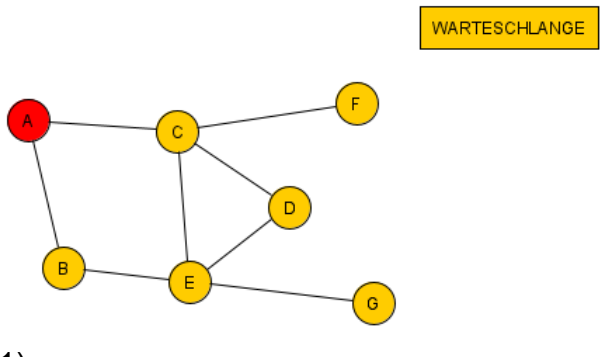
3.5 Breitensuche

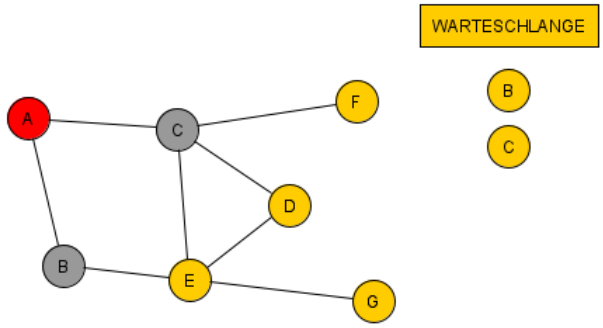
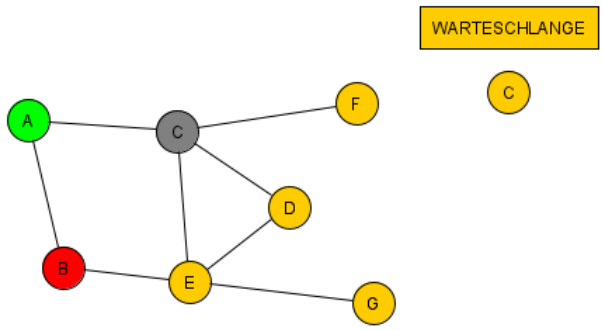
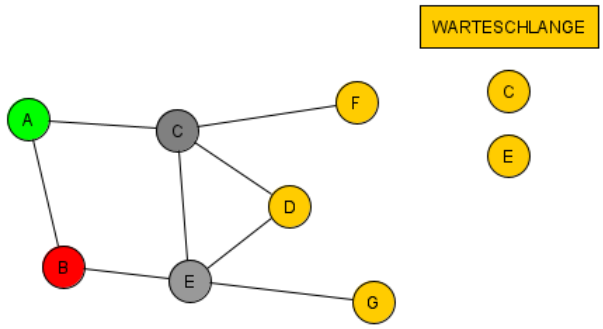
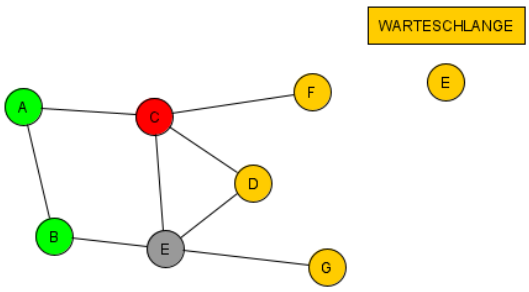
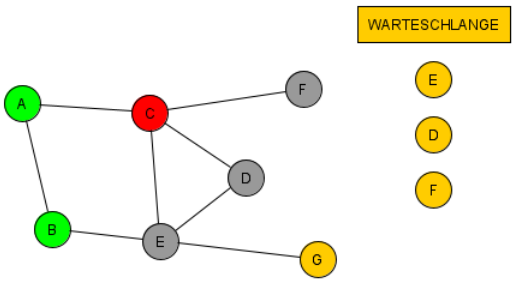
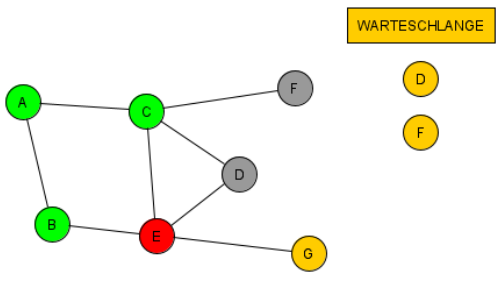
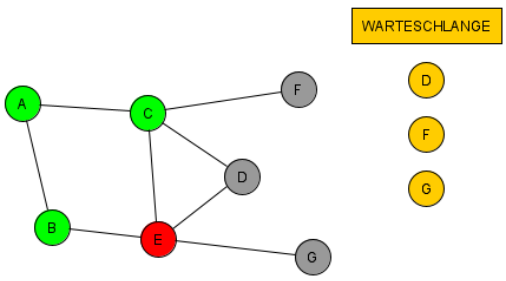
Um uns eine weitere Art einen Graph zu durchlaufen anzuschauen, betrachten wir folgende Situation:

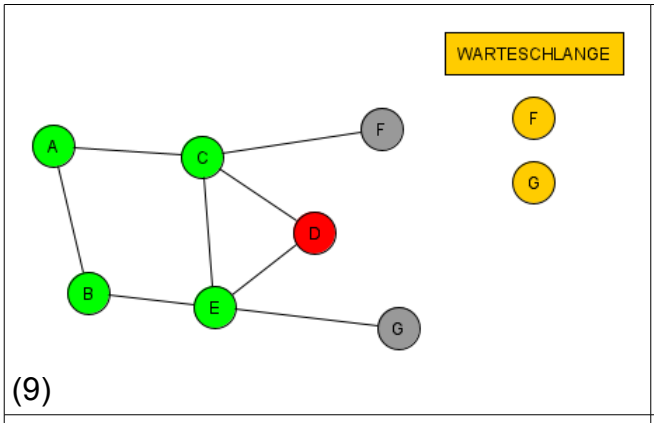
Anton, Bert, Carlos, Dominik, Elias, Franz und Georg sind zusammen in einem Wahlkurs an der Schule. Um Informationen untereinander zu verbreiten, schicken Sie sich gegenseitig Nachrichten. Dabei wollen Sie ein Schneeballsystem verwenden, in dem jeder die Informiert, deren Nummer er hat, da nicht jeder allen anderen seine Handynummer geben möchte.

Betrachtet man dieses Problem mit einem Graph, stellen die Knoten die einzelnen Personen und die Kanten jeweils wer wen anschreiben kann. Der Graph könnte dabei zum Beispiel so wie rechts aussehen.	

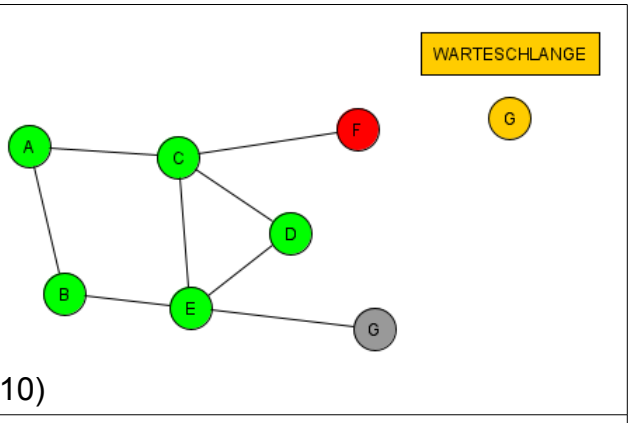
Ziel ist also wieder, dass alle Knoten durchlaufen werden (hier: jeder bekommt die Information). Aus der Tiefensuche wissen wir bereits, dass es sinnvoll ist ein Feld bereit zu halten, in dem gespeichert wird ob ein Knoten erledigt ist. Gehen wir das Beispiel einmal durch:

Legende: gelb = noch nicht besucht, rot = aktueller Standort, grün = erledigter Knoten	
Angenommen wir beginnen am Knoten Anton (A). Er müsste nun B und C anrufen und die Information weitergeben, da diese noch nicht besucht wurden.	 (1)

Es erscheint wieder sinnvoll, einfach der Reihe nach vorzugehen und zuerst B und dann C anzurufen.Die beiden werden in dieser Reihenfolge weiterarbeiten und somit in dieser Reihenfolge in die Warteschlange eingefügt.	 (2)
Wir arbeiten rekursiv weiter und gehen zu Knoten B. Dieser muss wieder überlegen, wer seine Nachbarn sind und diese in die Warteschlange einfügen.	 (3)
Diesmal ist nur E dazu gekommen. Wir können den Knoten B also abschließen und in der Reihenfolge der Warteschlange weiter machen. Das Prinzip setzt sich fort und wie man unten sieht, geht es immer gleich weiter.	 (4)
(5) 	(6) 
(7) 	(8) 

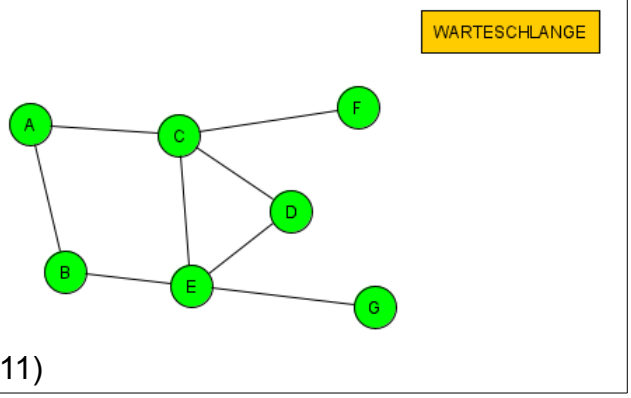


(9)



(10)

Wie wir sehen, brauchen wir im Gegensatz zur Tiefensuche keine Rücksprünge zu einem vorherigen Knoten, sondern arbeiten einfach die Warteschlange so lange ab, bis sie keine Elemente mehr enthält.



(11)

Der Algorithmus läuft also folgendermaßen:

1. Besuche den Startknoten
2. Besuchen: Markiere mich als erledigt
3. Speichere alle Nachbarn in einer Warteschlange (die noch nicht drin sind)
4. Überprüfe, ob die Warteschlange nicht leer ist.

    → Falls Ja: Besuche ersten Knoten der Warteschlange (→ weiter bei 2)

    → Falls Nein: Fertig

Das Prinzip der **Breitensuche** (engl. breath first search = BFS) ist, erst alle Nachbarn zu besuchen, bevor man von diesen aus tiefer in den Graph einsteigt.

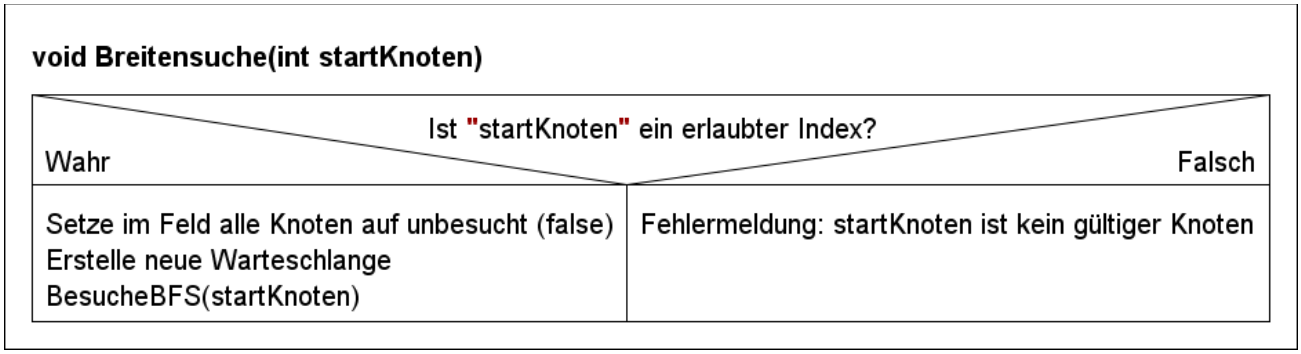
Umsetzung in Java

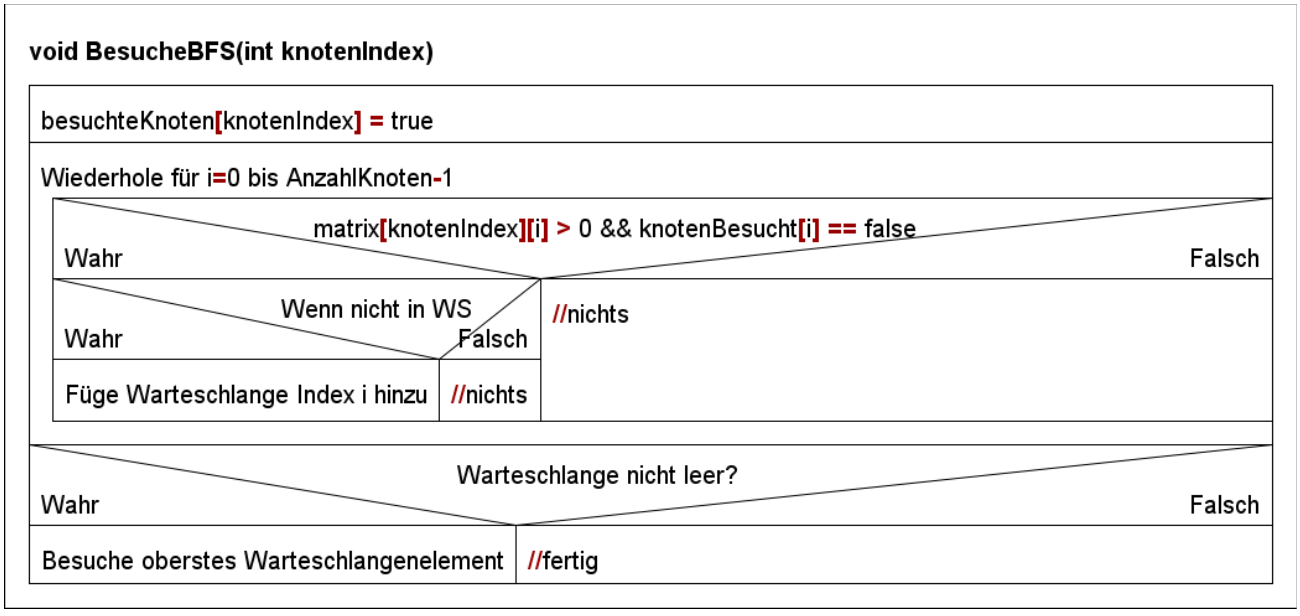
Klar ist, dass wir für die Markierungen, welcher Knoten bereits besucht wurde ein Feld mit Datentyp `boolean` brauchen. Da `boolean` ein primitiver Datentyp ist, müssen wir im Konstruktor nur angeben, wie groß das Feld ist, da primitive Datentypen automatisch mit einem Wert belegt werden. Bei `boolean` ist das tatsächlich der Wert `false`, was uns entgegenkommt.

Das Besuchen an sich können wir wieder mit einer Methode Besuchen lösen, da wir wieder wie bei der Tiefensuche immer wieder das gleiche tun. Diesmal könnten wir das ganze auch iterativ mit einer Wiederholung mit Ausgangsbedingung lösen, aber die Rekursion löst das kürzer und Eleganter für uns.

Wir erinnern uns, dass beim Graph die Adjazenzmatrix und alle Knoten in der Klasse GRAPH vorliegen und wir daher nur in dieser Klasse eine Methode einbauen müssen.

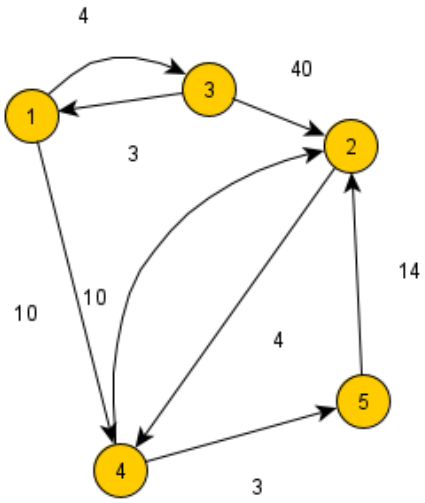
Überlegen wir uns die nötigen Struktogramme der Methoden `Breitensuche(int start)` und `BesucheBFS(int knotenIndex)`.





Aufgaben:

1) Überlegen Sie, in welcher Reihenfolge die Knoten im unserem bekannten Graph der Tiefensuche rechts besucht werden würden, wenn man den Graph ausgehend von Knoten 1 nach dem BFS Algorithmus besuchen würde.



2) Können Sie einen Graph angeben, in dem man mit dem BFS nicht alle Knoten besuchen würde?

3) Kopieren Sie die Vorlage Graph_mit_Adjazenzmatrix_BFS_Vorlage aus dem Klassenordner.

4) Machen Sie sich mit der Methode `Breitensuche` und der Klasse WARTESCHLANGE vertraut. Überlegen Sie vor allem, mit welchen Methoden sie Elemente in die WARTESCHLANGE einfügen und entfernen können und wie man überprüft, ob die WARTESCHLANGE leer ist. Die Vorlage hat diese Methode, die Klasse und das Feld `boolean[] erledigt` bereits implementiert.

5) Implementieren Sie den Algorithmus BFS in der Vorlage. Verwenden Sie die Methoden `Breitensuche` und `BesuchenBFS` in der Vorlage.

6) In manchen Anwendungen ist es wichtig, einen „Rahmen-GRAPH“ zu erstellen, der aus dem GRAPH besteht, der nur durch das besuchen nach BFS entsteht. Er enthält alle KNOTEN, aber nur die KANTEN, die beim BFS benutzt wurde. Implementieren Sie eine Methode `BFSGraph` in der Klasse GRAPH, die aus dem existierenden Graph den Rahmen-GRAPH erstellt und diesen zurückgibt.