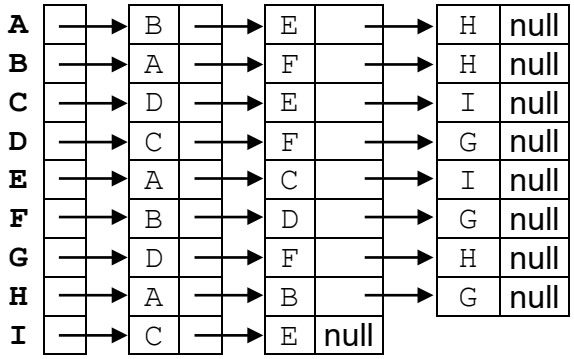
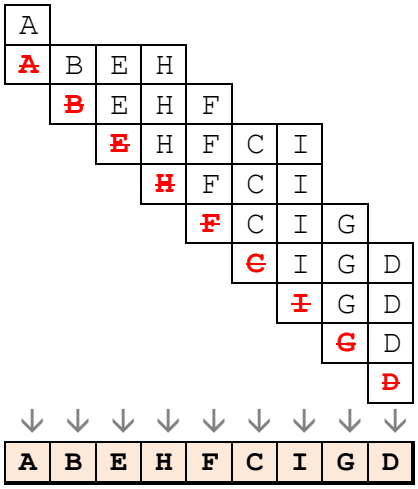
VL12, Lösung 1

a) Aus dem Graphen ergibt sich folgende Listenstruktur:

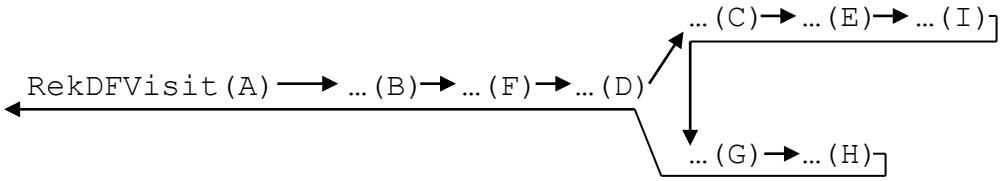


b) A, B, E, H, F, C, I, G, D

Die Reihenfolge ergibt sich aufgrund der verwendeten **Queue**, in die die noch zu besuchenden Knoten eingetragen werden:



c) Der Algorithmus `RekDfs` durchläuft die Liste der Nachbarn entsprechend der Reihenfolge in der Listenstruktur, d.h. hier alphabetisch. Daraus ergibt sich bei der Traversierung die Knotenfolge A, B, F, D, C, E, I, G, H.

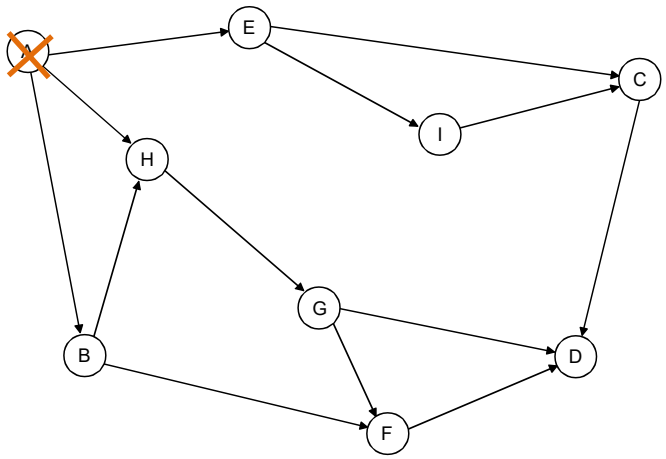


VL12, Lösung 2

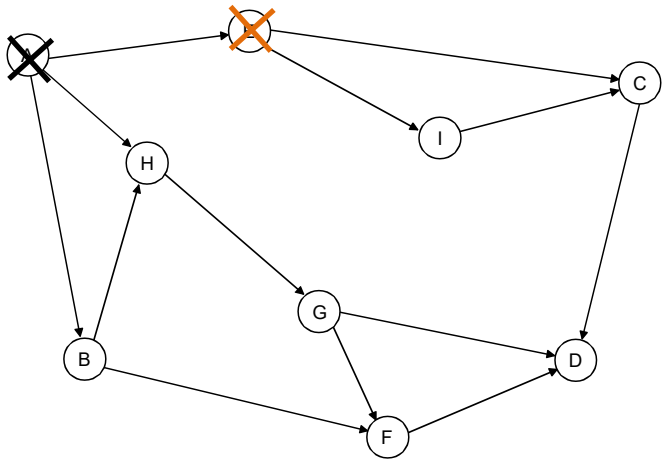
Es sind mehrere Lösungen möglich, wie z.B. `A`, `E`, `I`, `C`, `B`, `H`, `G`, `F`, `D`.

Beispiel für den Ablauf des Algorithmus:

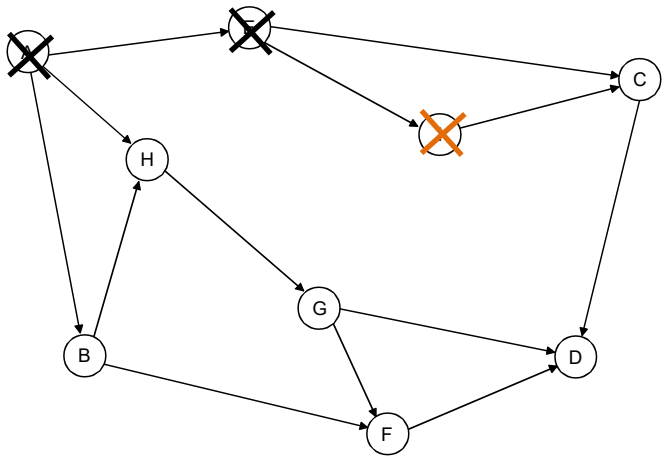
Wähle einen Knoten ohne Vorgänger: nur A möglich.



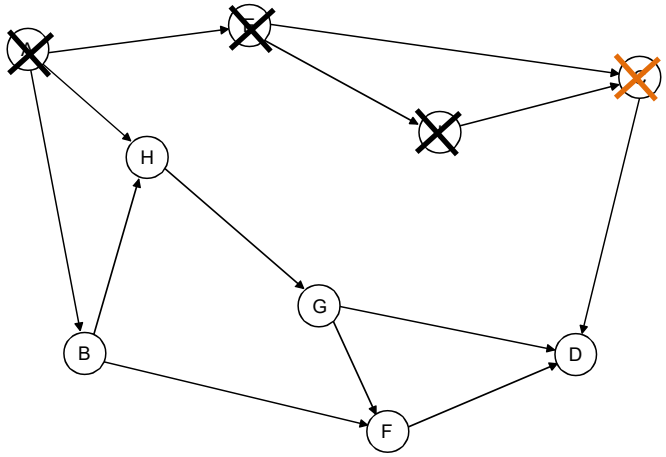
Wähle einen Knoten ohne Vorgänger: B oder E möglich. Wähle E.



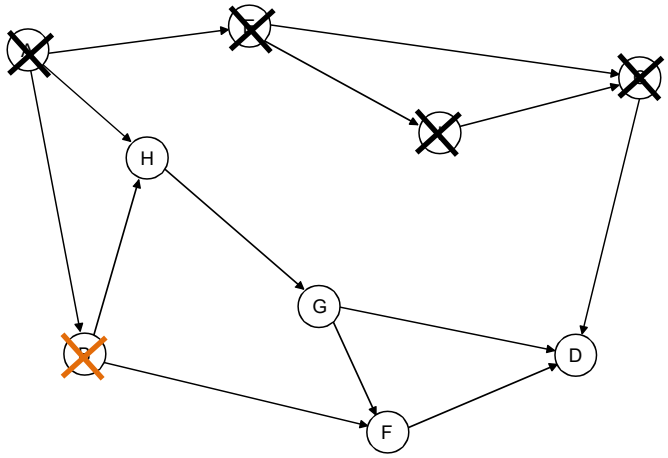
Wähle einen Knoten ohne Vorgänger: B oder I möglich. Wähle I.



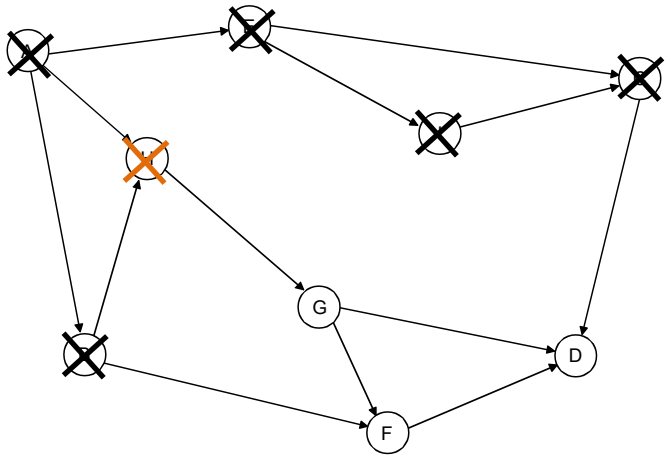
Wähle einen Knoten ohne Vorgänger: B oder C möglich. Wähle C.



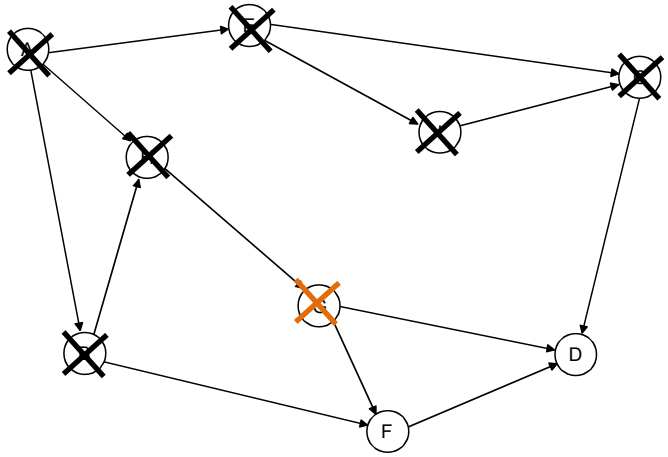
Wähle einen Knoten ohne Vorgänger: nur B möglich.



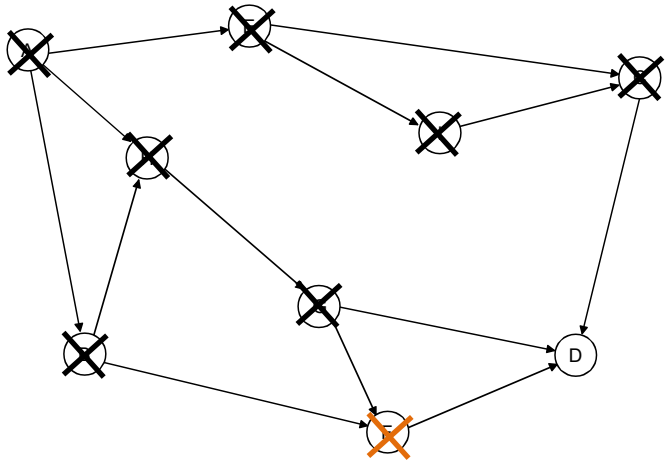
Wähle einen Knoten ohne Vorgänger: nur H möglich.



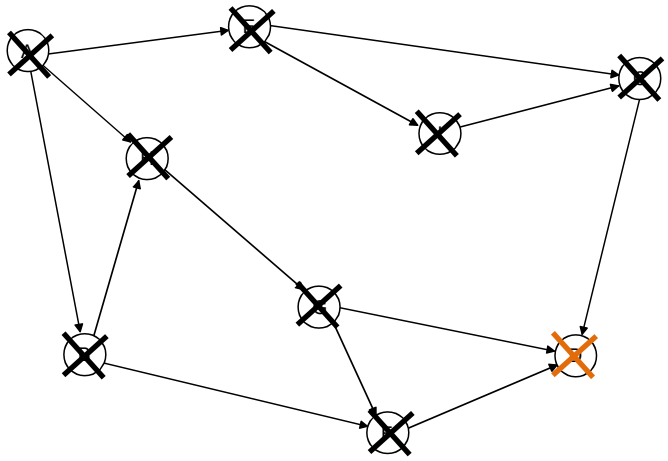
Wähle einen Knoten ohne Vorgänger: nur G möglich.



Wähle einen Knoten ohne Vorgänger: nur F möglich.



Wähle einen Knoten ohne Vorgänger: nur D möglich.



Fertig.

VL12, Lösung 3

Es wird eine Markierung (besucht) für jeden Knoten initialisiert. Es werden der Reihe nach alle Knoten durchgegangen. Wird ein noch nicht besuchter Knoten gefunden, so beginnt die Ausgabe einer neuen Zusammenhangskomponente. Von diesem Knoten aus werden nun per Breiten- oder Tiefensuche alle erreichbaren Knoten besucht. **Eine etwaige Kantenrichtung** (gerichteter Graph) **wird hierbei ausnahmsweise ignoriert!**

Pseudocode:

```
bestimmeZusammenhangskomponeten()
   initialisiere Knotenmarkierungen // keine Knoten markiert
   for jeder Knoten k von G do
      if k nicht markiert then
         // neue Zusammenhangskomponente
         rekDfs(k)                  // markieren und ausgeben
      end if
   end for
```

Es existieren die Zusammenhangskomponenten (A, B, C, F), (D, E) und (G).

VL12, Lösung 4

```
private boolean isKante(int k1, int k2)
{
     return matrix[k1][k2] | matrix[k2][k1];
}

// Tiefensuche
private void rekDfs(int k)
{
     besucht[k] = true;
     System.out.print(knoten[k].getName() + " ");

     for (int a = 0; a < KNOTENZAHL; a++)
          if (isKante(k, a))
               if (!besucht[a])
                    rekDfs(a);
}
```

```
public void zusammenhangskomponenten()
{
      besucht = new boolean[KNOTENZAHL];

      int anzZusammenhangskomponenten = 0;
      for (int a = 0; a < KNOTENZAHL; a++)
            if (!besucht[a])
            {
                  System.out.print("Zusammenhangskomponente " +
                              ++anzZusammenhangskomponenten + ": ");
                  rekDfs(a);
                  System.out.println();
            }
}

// Breitensuche
public void bfsIterativ(final int start)
{
      boolean[] besucht = new boolean[KNOTENZAHL];
      Deque<Integer> q = new LinkedList<Integer>();

      q.addLast(start);
      besucht[start] = true;

      while (!q.isEmpty())
      {
            int k = q.removeFirst();
            System.out.print(knoten[k].getName() + " ");

            for (int a = 0; a < KNOTENZAHL; a++)
                  if (isKante(k, a))
                        if (!besucht[a])
                        {
                              q.addLast(a);
                              besucht[a] = true;
                        }
      }
      System.out.println();
}
```

Ein Java-Programm zur Lösung der Aufgaben 4 und finden Sie zusätzlich in der Datei ML12-Aufgabe_4_5.zip.