Prof. Dr. E. Ábrahám

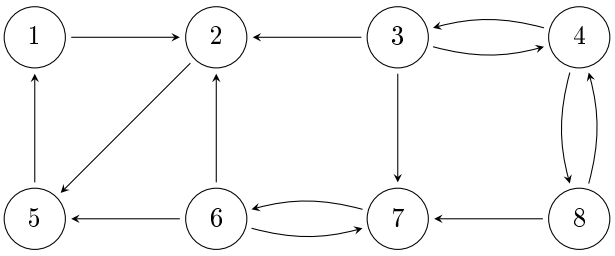
F. Corzilius, S. Schupp, T. Ströder

Allgemeine Hinweise:

- Die **Hausaufgaben** sollen in Gruppen von je **2 bis 3 Studierenden** aus der **gleichen Kleingruppenübung (Tutorium)** bearbeitet werden. **Namen und Matrikelnummern** der Studierenden sind auf jedes Blatt der Abgabe zu schreiben. **Heften bzw. tackern Sie die Blätter!**
- Die **Nummer der Übungsgruppe** muss **links oben** auf das **erste Blatt** der Abgabe geschrieben werden. Notieren Sie die Gruppennummer gut sichtbar, damit wir besser sortieren können.
- Die Lösungen müssen **bis Montag, den 07.07.2014 um 9:00 Uhr** in den entsprechenden Übungskasten eingeworfen werden. Sie finden die Kästen am Eingang Halifaxstr. des Informatikzentrums (Ahornstr. 55). Alternativ können Sie die Lösungen auch in Ihrem Tutorium vor der Abgabefrist direkt bei Ihrer Tutorin/Ihrem Tutor abgeben.

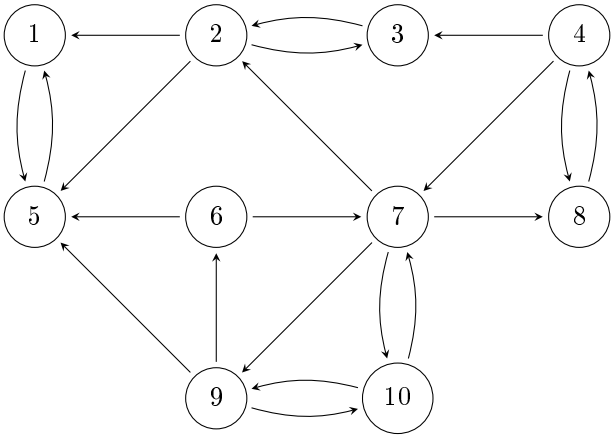
Tutoraufgabe 1 (SCC):

Wenden Sie Sharir's Algorithmus an (siehe Folien zur Vorlesung) um die starken Zusammenhangskomponenten des folgenden Graphen zu finden. Geben Sie das Array `color` und den Stack `S` nach jeder Schleifeniteration der ersten und zweiten Phase (also nach Zeile 17 und nach Zeile 22) an, falls `DFS1` bzw. `DFS2` ausgeführt wurde. Geben Sie zudem das Array `scc` nach jeder Schleifeniteration der zweiten Phase (also nach Zeile 22) an, falls `DFS2` ausgeführt wurde. Nehmen Sie hierbei an, dass `scc` initial mit Nullen gefüllt ist und der Knoten mit Schlüssel i in der Adjazenzliste den $(i-1)$-ten Eintrag hat, also der Knoten mit Schlüssel 1 vom Algorithmus als erstes berücksichtig wird usw.



Aufgabe 2 (SCC): (5 Punkte)

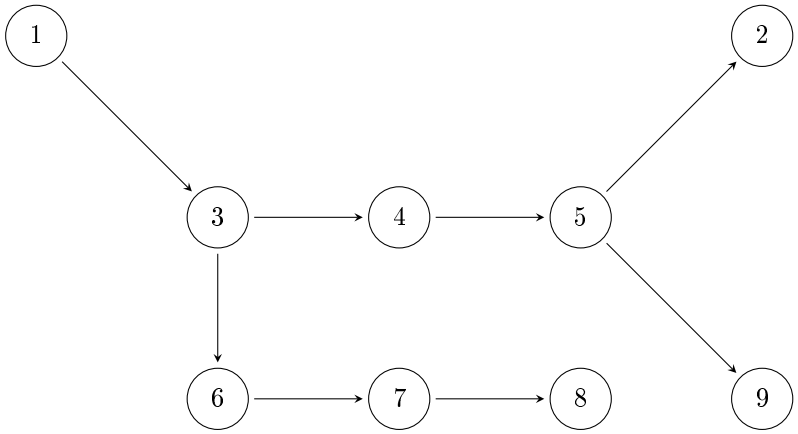
Wenden Sie Sharir's Algorithmus an (siehe Folien zur Vorlesung) um die starken Zusammenhangskomponenten des folgenden Graphen zu finden. Geben Sie das Array `color` und den Stack `S` nach jeder Schleifeniteration der ersten und zweiten Phase (also nach Zeile 17 und nach Zeile 22) an, falls `DFS1` bzw. `DFS2` ausgeführt wurde. Geben Sie zudem das Array `scc` nach jeder Schleifeniteration der zweiten Phase (also nach Zeile 22) an, falls `DFS2` ausgeführt wurde. Nehmen Sie hierbei an, dass `scc` initial mit Nullen gefüllt ist und der Knoten mit Schlüssel i in der Adjazenzliste den $(i-1)$-ten Eintrag hat, also der Knoten mit Schlüssel 1 vom Algorithmus als erstes berücksichtig wird usw.



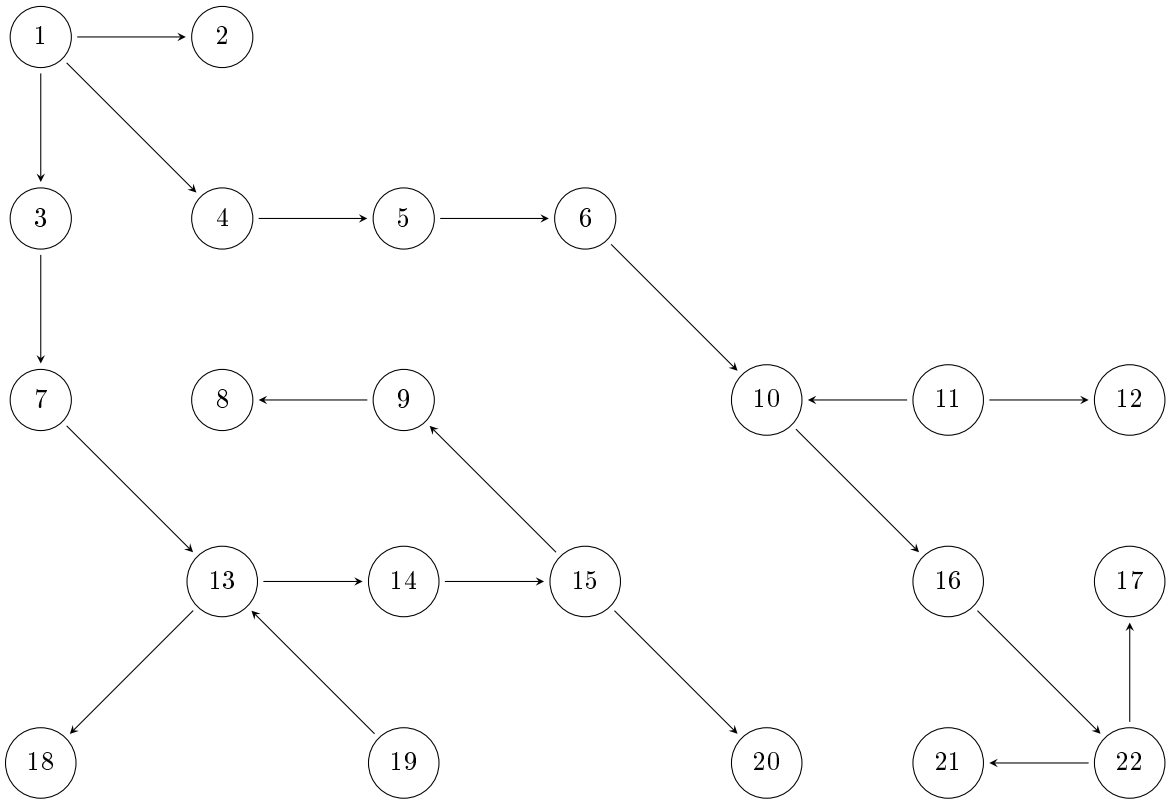
Tutoraufgabe 3 (Topologisches Sortieren):

Geben Sie eine topologische Sortierung des folgenden Graphen an. Dafür reicht es, eine geordnete Liste der Knoten mit dem dazugehörigen Topologiewert in Klammern anzugeben. Die Tiefensuche berücksichtigt bei mehreren Kindern diese in aufsteigender Reihenfolge (ihrer Schlüssel). Des Weiteren ist jedes Array, welches Knoten beinhaltet, aufsteigend nach deren Schlüsseln sortiert. Beachten Sie, dass der Knoten mit Schlüssel i in der Adjazenzliste den $(i-1)$-ten Eintrag hat, also der Knoten mit Schlüssel 1 vom Algorithmus als erstes berücksichtig wird usw.

a)



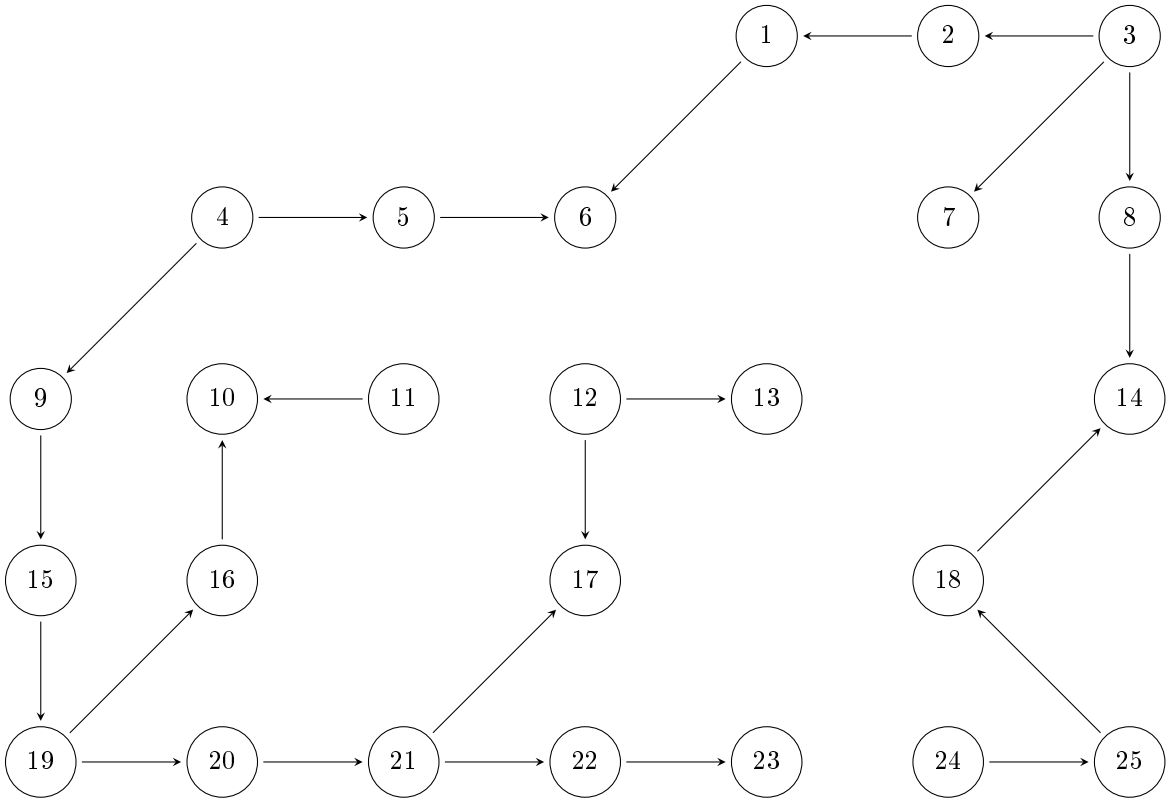
b)



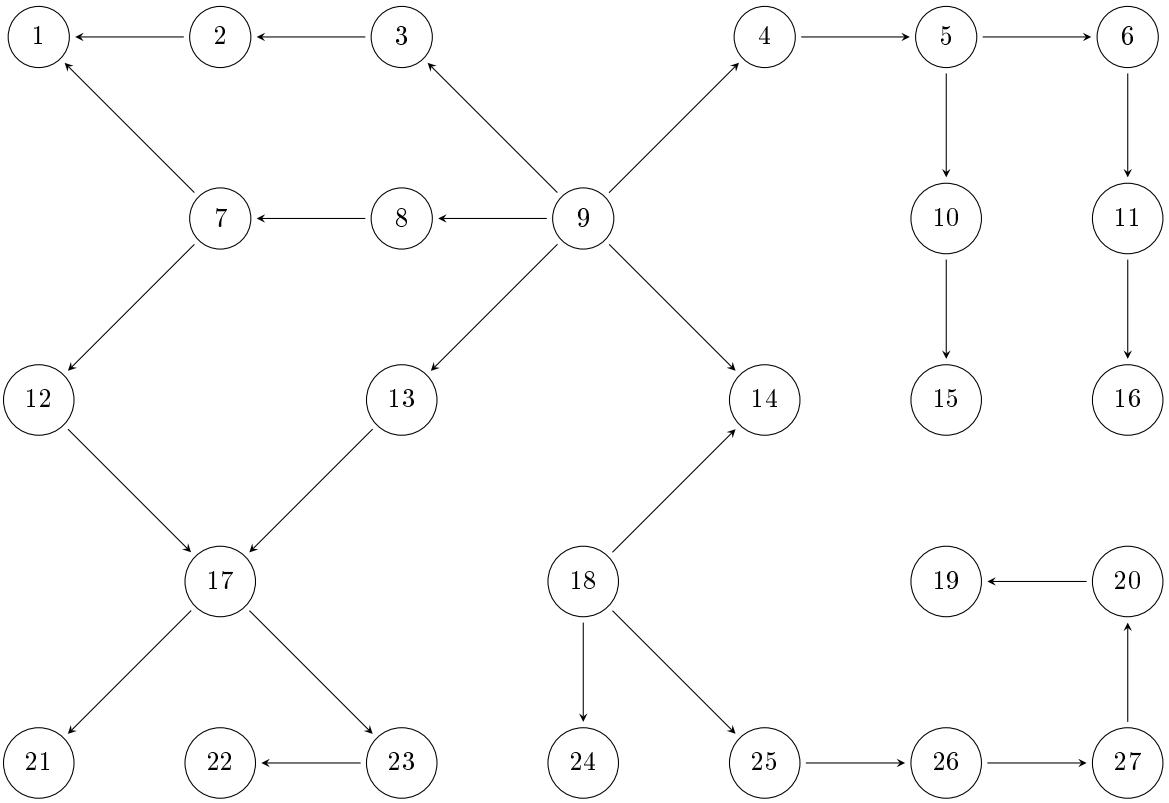
Aufgabe 4 (Topologisches Sortieren): (3 + 3 = 6 Punkte)

Geben Sie eine topologische Sortierung des folgenden Graphen an. Dafür reicht es, eine geordnete Liste der Knoten mit dem dazugehörigen Topologiewert in Klammern anzugeben. Die Tiefensuche berücksichtigt bei mehreren Kindern diese in aufsteigender Reihenfolge (ihrer Schlüssel). Des Weiteren ist jedes Array, welches Knoten beinhaltet aufsteigend nach deren Schlüsseln sortiert. Beachten Sie, dass der Knoten mit Schlüssel i in der Adjazenzliste den $(i-1)$-ten Eintrag hat, also der Knoten mit Schlüssel 1 vom Algorithmus als erstes berücksichtig wird usw.

a)



b)

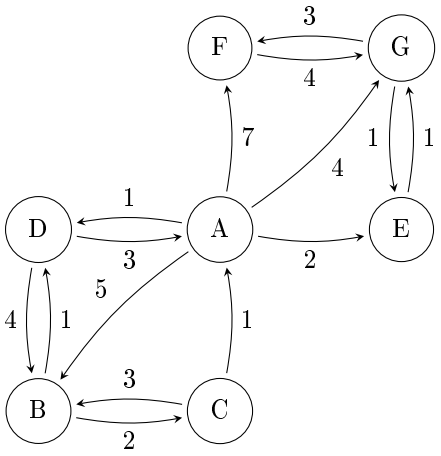


Tutoraufgabe 5 (Dijkstra):

Der Dijkstra Algorithmus zur Berechnung kürzester Entfernungen von einem Startknoten zu allen anderen Knoten arbeitet gemäß dem folgenden Pseudo-Code:

```
//Inp.: gerichteter gewichteter Graph mit n Knoten, Startknoten
void dijkstra(int adj[n], int n, int start) {
    for (int i = 0; i < n; i++)
        key[i] = inf;
    key[start] = 0; //start ist einziger Randknoten mit Kosten 0
    Q = {0,...,n-1}; //Q enthält alle ungesehenen Knoten und
                     //alle Randknoten
    while (Q not empty) {
        //verschiebe den billigsten Randknoten v in den Baum:
        //extractMin(Q) bestimmt ein Element e aus Q mit minimalem
        //Schlüssel key[e], entfernt e aus Q und gibt e zurück
        v = extractMin(Q);
        // aktualisiere Kosten für Randnnoten
        for each ((u,w) in adj[v])
            if (u in Q and key[v] + w < key[u])
                key[u] = key[v] + w;
    }
}
```

Betrachten Sie den folgenden Graphen:

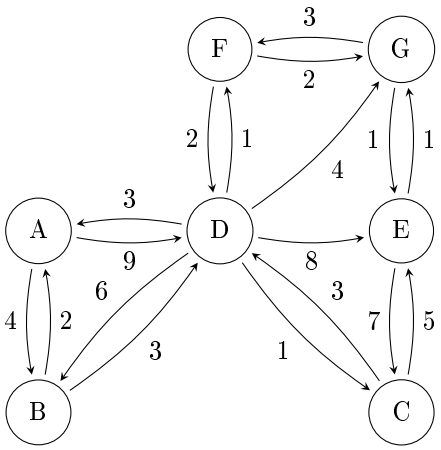


Führen Sie den Dijkstra Algorithmus auf diesem Graphen mit dem Startknoten A aus. Füllen Sie dazu die nachfolgende Tabelle aus, indem Sie den Wert von `v` und `key` nach jeder Iteration der `while`-Schleife eintragen:

v	A					
key[A]						
key[B]						
key[C]						
key[D]						
key[E]						
key[F]						
key[G]						

Aufgabe 6 (Dijkstra): (6 Punkte)

Betrachten Sie den folgenden Graphen:



Führen Sie den Dijkstra Algorithmus auf diesem Graphen mit dem Startknoten A aus. Füllen Sie dazu die nachfolgende Tabelle aus, indem Sie den Wert von `v` und `key` nach jeder Iteration der `while`-Schleife eintragen:

v	A					
key[A]						
key[B]						
key[C]						
key[D]						
key[E]						
key[F]						
key[G]						