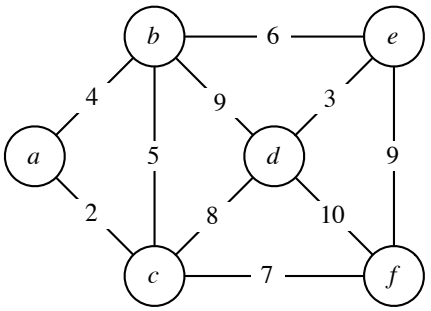
**Aufgabe 10:** (8 Punkte)

Betrachten Sie den nebenstehenden ungerichteten, gewichteten Graphen $G = (V, E)$ mit Gewichtsfunktion $c : E \to \mathbb{Z}$.




Bestimmen Sie für diesen Graphen mit Hilfe des Algorithmus von Prim einen minimalen Spannbaum. Wählen Sie $a$ als Startknoten und geben Sie *vor jeder* Auswahl einer Kante für den Spannbaum an,

- wie der Aufbau der verwendeten Heap-Datenstruktur zu diesem Zeitpunkt aussieht (Heap-Updates!) und
- welche Kante in diesem Schritt ausgewählt wird.

Geben Sie den resultierenden minimalen Spannbaum und seine Kosten an.

**Aufgabe 11:** (6 Punkte)

Betrachten Sie einen ungerichteten, gewichteten Graphen $G = (V, E)$ mit Gewichtsfunktion $c : E \to \mathbb{Z}$. Zeigen oder widerlegen Sie dazu jeweils:

(a) Falls in $G$ alle Kantengewichte unterschiedlich sind, dann besitzt $G$ einen eindeutig bestimmten minimalen Spannbaum.

(b) Falls $G$ einen eindeutig bestimmten minimalen Spannbaum besitzt, so sind alle Kantengewichte paarweise verschieden.

(c) Es gibt einen Graphen $G$, bei dem eine Kante maximalen Gewichts in jedem minimalen Spannbaum von $G$ enthalten ist.

**Programmieraufgabe P3:** (10 Punkte)

Implementieren Sie den Algorithmus von Prim in zwei Varianten: einmal mit und einmal ohne die Nutzung eines Heaps. Testen Sie beide Implementierungen auf den Graphen `prim_01.gra`, `prim_02.gra` und `prim_03.gra` (siehe studIP) sowie auf einigen selbst generierten Instanzen und vergleichen Sie die Laufzeiten.

Die Klasse soll den Namen `Prim` haben und die Methode `public static RenderableGraph minimumSpanningTree(RenderableGraph g)` implementieren. Den erzeugten `RenderableGraph` können Sie der Methode `RenderGraph.renderGraph(Renderable Graph graph, String filename)` übergeben, um den Graphen zeichnen zu lassen.

Ihre Ausgabe sollte beinhalten, in welcher Reihenfolge welche Kanten ausgewählt werden und welche Kosten der berechnete minimale Spannbaum besitzt.