Prof. Dr. Leif Kobbelt
Stefan Dollase, Ira Fesefeldt, Alexandra Heuschling, Gregor Kobsik

Übung 10

Tutoraufgabe 1 (Einfach Verbundene Graphen):

a) Sei G ein gerichteter Graph. Wir nennen G *einfach verbunden*, wenn es zwischen jedem Knotenpaar $u, v \in V$ höchstens einen einfachen Pfad zwischen u und v gibt.

Geben Sie einen möglichst effizienten Algorithmus an, um herauszufinden ob ein gerichteter Graph einfach verbunden ist. Begründen Sie die Korrektheit und die Laufzeit Ihres Algorithmus.

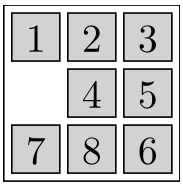
b) Sei G ein gerichteter azyklischer Graph. Geben Sie einen möglichst effizienten Algorithmus an, um herauszufinden, ob G einfach verbunden ist. Begründen Sie die Korrektheit und die Laufzeit Ihres Algorithmus.

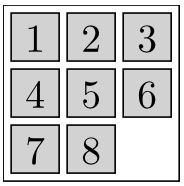
c) Ein einfacher Kreis $v_0 \dots v_{k-1} v_0$ ist ein Kreis, für den der Pfad $v_0 \dots v_{k-1}$ einfach ist.

Geben Sie einen möglichst effizienten Algorithmus an, um herauszufinden ob ein ungerichteter Graph einen einfachen Kreis enthält. Begründen Sie die Korrektheit und die Laufzeit Ihres Algorithmus.

Tutoraufgabe 2 (Neunerschiebepuzzle):

Das „Neunerschiebepuzzle“ besteht aus acht beweglichen Feldern in einer 3×3-Matrix. Jeweils eine der neun Positionen ist frei und ein an die freie Position angrenzendes Feld kann in diese hineingeschoben werden. Dies nennen wir einen „Zug“.





Ziel des Spiels ist es, die rechts gezeigte Position zu erreichen. Wir können uns nun einen ungerichteten Graphen vorstellen, dessen Knoten die Positionen dieses Spiels sind. Zwei Positionen sind durch eine Kante verbunden, wenn ein Zug sie ineinander überführt.
Wir können eine Breitensuche auf diesen Graphen beginnen, ohne ihn vorher komplett zu konstruieren. Führen Sie eine solche Breitensuche auf dem links gezeigten Startknoten aus und brechen Sie sie ab, sobald die Zielposition erscheint.

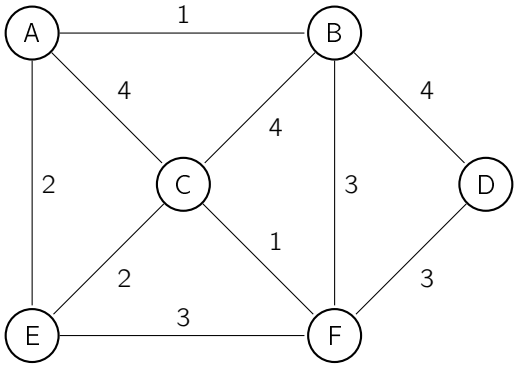
a) Zeichnen Sie den bis dahin entstandenen Breitensuchbaum auf.

b) Können Sie jetzt eine kürzestmögliche Lösung des Rätsels aus diesem Baum ablesen?

c) Ist es in dieser und ähnlichen Situationen Ihrer Meinung nach besser eine Tiefen- oder eine Breitensuche durchzuführen?

Tutoraufgabe 3 (Prim(-Jarnik-Dijkstra) Algorithmus und Heaps):

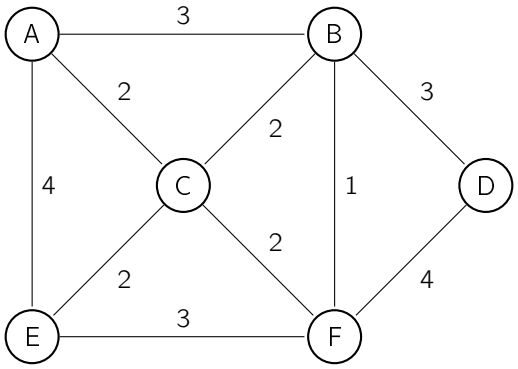
Gegeben ist folgender Graph G:



Führen Sie Prims Algorithmus auf den Graphen G mit Startknoten A aus, um einen Minimalen Spannbaum zu berechnen. Geben Sie dabei nach jeder Operation auf dem Heap den neuen Zustand des Heaps, sowie die gespeicherten Kosten jedes Knotens an. Es ist nicht nötig, alle Zwischenschritte beim initialen Erstellen des Heaps anzugeben. Gehen Sie außerdem davon aus, dass über Knoten stets in alphabetischer Reihenfolge iteriert wird. Sie dürfen den Heap sowohl als Baum, als auch als Array angeben. Geben Sie außerdem den resultierenden Minimalen Spannbaum an.

Aufgabe 4 (Kruskals Algorithmus und Union-Find): 10 Punkte

Gegeben ist folgender Graph G:



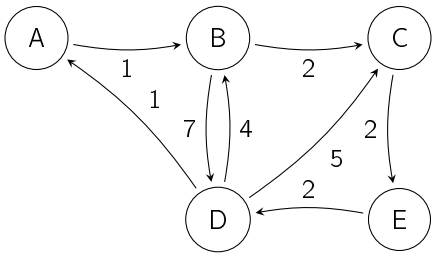
Führen Sie Kruskals Algorithmus auf den Graphen G aus, um einen Minimalen Spannbaum zu berechnen. Geben Sie dabei nach jeder Operation auf der Union-Find Struktur die entsprechende Operationen und den neuen Zustand der Union-Find Struktur an falls diese sich verändert hat. Dabei soll die Union-Operation bei **gleicher Höhe der Wurzeln immer die Wurzel des zweiten Parameters** als neue Wurzel wählen. Benutzen Sie hierfür sowohl Pfadkompression als auch Höhenbalancierung. Es ist nicht nötig, alle Zwischenschritte beim Ausführen der MakeSet Operationen anzugeben. Benutzen Sie die folgende Sortierung der Kanten:

$$(B,F),(A,C),(B,C),(C,E),(C,F),(A,B),(B,D),(E,F),(A,E),(D,F)$$

Geben Sie außerdem den resultierenden Minimalen Spannbaum an.

Aufgabe 5 (Floyd-Warshall Algorithmus): 10 Punkte

Betrachten Sie den folgenden Graphen:



Führen Sie den *Algorithmus von Floyd* auf diesem Graphen aus. Geben Sie dazu nach jedem Durchlauf der äußeren Schleife die aktuellen Entfernungen in einer Tabelle an. Die erste Tabelle enthält bereits die Adjazenzmatrix nach Bildung der reflexiven Hülle. Der Eintrag in der Zeile i und Spalte j ist also ∞, falls es keine Kante vom Knoten der Zeile i zu dem Knoten der Spalte j gibt, und sonst das Gewicht dieser Kante. Beachten Sie, dass in der reflexiven Hülle jeder Knoten eine Kante mit Gewicht 0 zu sich selbst hat.

①	A	B	C	D	E
A	0	1	∞	∞	∞
B	∞	0	2	7	∞
C	∞	∞	0	∞	2
D	1	4	5	0	∞
E	∞	∞	∞	2	0

②	A	B	C	D	E
A					
B					
C					
D					
E					

③	A	B	C	D	E
A					
B					
C					
D					
E					

④	A	B	C	D	E
A					
B					
C					
D					
E					

⑤	A	B	C	D	E
A					
B					
C					
D					
E					

⑥	A	B	C	D	E
A					
B					
C					
D					
E					

Aufgabe 6 (Programmierung in Python - Graphalgorithmen):

2 + 3 + 5 + 6 + 4 = 20 Punkte

Bearbeiten Sie die Python Programmieraufgaben. In dieser praktischen Aufgabe werden Sie sich mit Graphalgorithmen auseindersetzen. Diese Aufgabe dient dazu einige Konzepte der Vorlesung zu wiederholen und zu vertiefen. Zum Bearbeiten der Programmieraufgabe können Sie einfach den Anweisungen des Notebooks *blatt10-python.ipynb* folgen. Das Notebook steht in der .zip-Datei zum Übungsblatt im Lernraum zur Vergügung.
Ihre Implementierung soll immer nach dem # *YOUR CODE HERE* Statement kommen. Ändern Sie keine weiteren Zellen.
Laden Sie spätestens bis zur Deadline dieses Übungsblatts auch Ihre Lösung der Programmieraufgabe im Lernraum hoch. Die Lösung des Übungsblatts und die Lösung der Programmieraufgabe muss im Lernraum an derselben Stelle hochgeladen werden. Die Lösung des Übungsblatts muss dazu als .pdf-Datei hochgeladen werden. Die Lösung der Programmieraufgabe muss als .ipynb-Datei hochgeladen werden.

Übersicht der zu bearbeitenden Aufgaben:

a) Implementierung von Dijkstra

- initialize_single_source()
- relax()
- dijkstra()

b) Labyrinthe: Generieren und Lösen

- generate_maze()
- solve_maze()