Graphenalgorithmen: Blatt 5

Lukas Kalbertodt, Elena Resch, Mirko Wagner

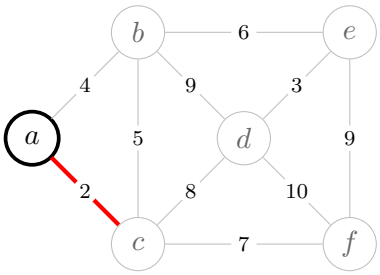
21. Mai 2015

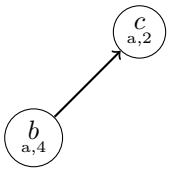
Aufgabe 10:

Hinweis: Es wird angenommen, dass mit „Heap“ ein „BinaryHeap“ gemeint ist.

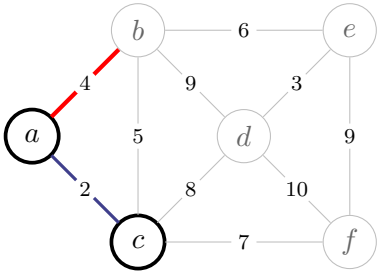
Legende für den Graphen: Dick schwarz umkreiste Knoten sind im Spannbaum enthalten. Die dicke rote Kante wird in dem Schritt als neue Kante dem Spannbaum hinzugefügt. Blaue Kanten befinden sich bereits im Spannbaum.

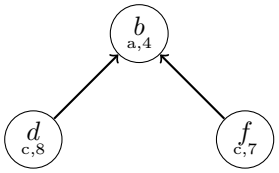
Legende für den BinaryHeap: Der dickere Buchstabe in jedem Knoten des Heaps steht für den Knoten im Graphen. In der zweiten Zeile steht zuerst der Predecessor und dann die Kosten.



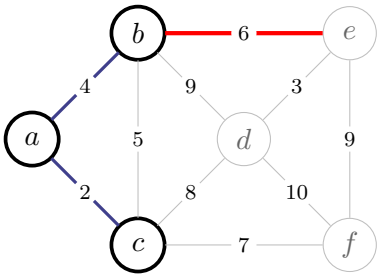


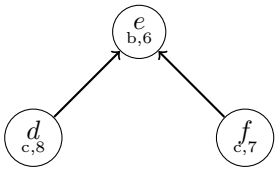
- `insert(b, a, 4)`
- `insert(c, a, 2)`



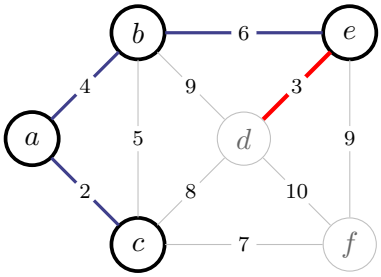


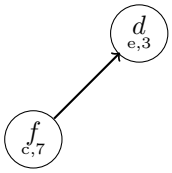
- `remove(c)`
- `insert(d, c, 8)`
- `insert(f, c, 7)`



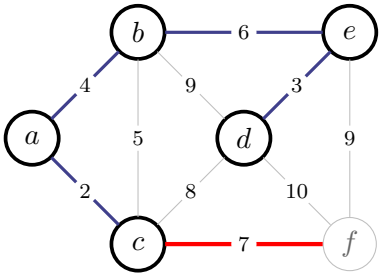


- `remove(b)`
- `insert(e, b, 6)`

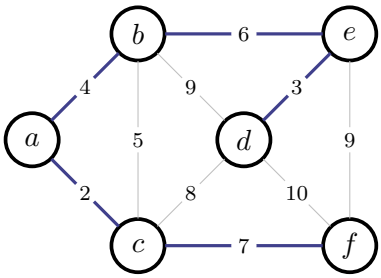




- `remove(e)`
- `update(d, e, 3)`



- `remove(d)`



- `remove(f)`

Kosten des MST: $c = 22$

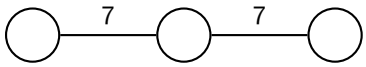
Aufgabe 11:

(a) Seien alle Kantengewichte in G paarweise unterschiedlich. Gegeben sei T, ein minimaler Spannbaum von G mit dem Wert v. Angenommen es gibt einen weiteren Spannbaum $T^* \neq T$ von G mit dem Wert v^*, wobei $v^* = v$ (also ist T^* auch minimal).
Wir wählen nun eine Kante $e \in T \setminus T^*$ und löschen sie aus T. T ist jetzt ein Wald mit zwei Komponenten/Bäumen. T^* besitzt eine Kante e^*, die diese beiden Komponenten in T verbindet. Nach Vorraussetzung ist $c(e) \neq c(e^*)$, somit können genau zwei Fälle auftreten:

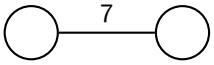
1. $c(e) > c(e^*)$: Wenn wir in T die Kante e durch $e*$ ersetzen würden, hätten wir wieder einen Spannbaum, aber mit geringeren Kosten. Weder T noch T^* wären also minimale Spannbäume: Wiederspruch!
2. $c(e) < c(e^*)$: Analog.

□

(b) Die Behauptung ist falsch. Im folgenden trivialen Gegenbeispiel ist der Spannbaum eindeutig bestimmt, aber es gibt zwei gleiche Kantenkosten:



(c) In allen Graphen, die in zwei Komponenten zerfallen, wenn man die teuerste Kante löscht, ist die teuerste Kante offentsichtlich auch Teil jedes Spannbaums. Minimalbeispiel:



Programmieraufgabe P3:

Für kleine Instanzen von Graphen sind in den Implementierungen der Methoden keine bzw. kaum Unterschiede vorhanden. Werden jedoch die Instanzen größer, beispielsweise bei vollständigen Graphen mit $n \geq 20$, so werden die Unterschiede der Laufzeiten deutlich:

	Laufzeit (sec)	
n	heap	lists
10	0.003	0.004
50	0.026	0.019
100	0.034	0.035
500	0.608	1.029
1000	4.302	7.548
2000	32.1	60.485

Die Erklärung liegt in der Menge der Kanten. Ein Heap muss in jedem nächstbilligsten Knoten die neu dazugekommenen Kanten zum Heap zufügen oder diese updaten. Die von uns implementierte Methode mit den Listen übernimmt diese Idee, aber da Listenlaufzeiten langsamer sind, ist auch die Methode langsamer.