Algorithmen und Datenstrukturen: Übung 6

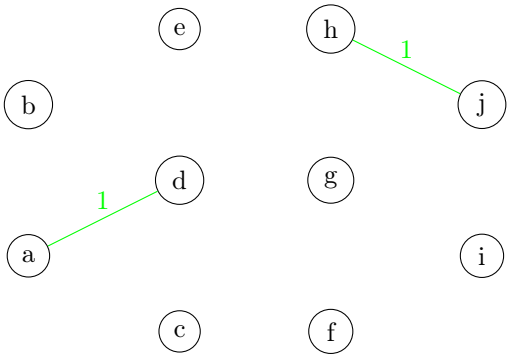
Tanja Zast, Alexander Waldenmaier

16. Dezember 2020

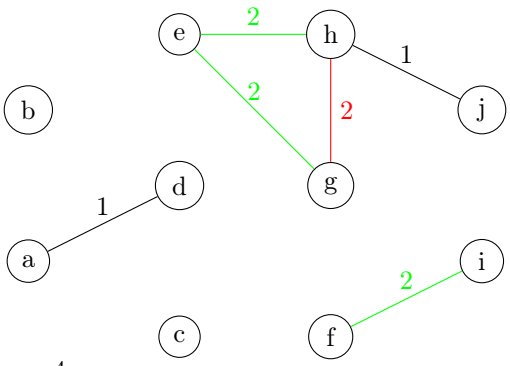
Aufgabe 6.1

Im Folgenden wir der Kruskal-Algorithmus sukkzesive für aufsteigende Kantengewichte durchgeführt. Für jedendes Gewicht werden akzeptierte Kanten grün eingezeichnet, abgelehnte Kanten rot und Kanten von vorigen Gewichten schwarz.

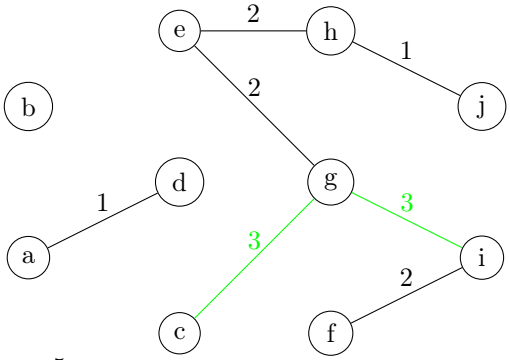
$g = 1$:



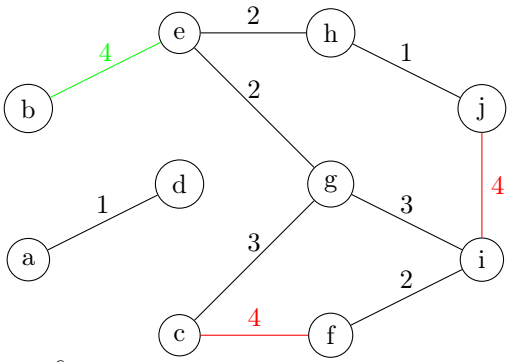
$g = 2$:



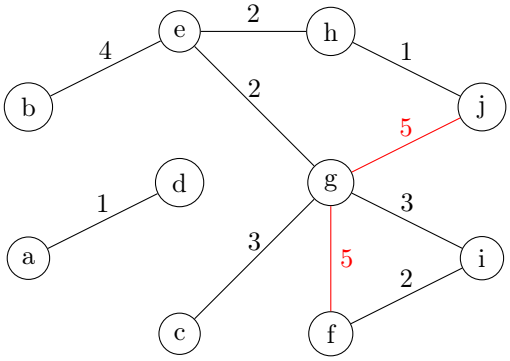
$g = 3$:



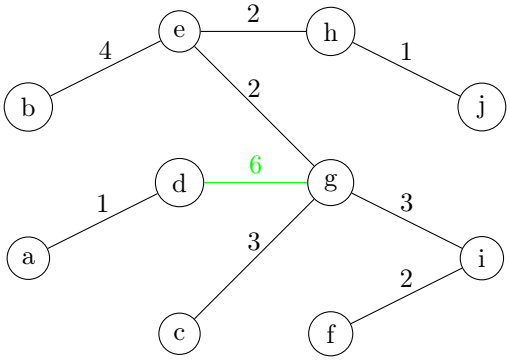
$g = 4$:



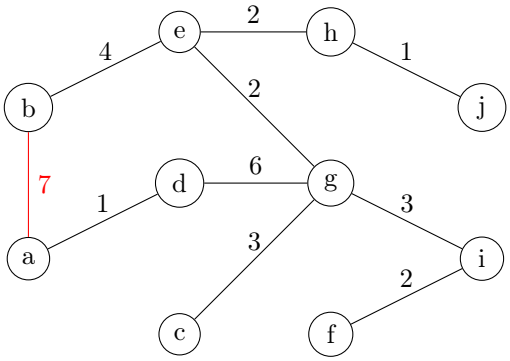
$g = 5$:



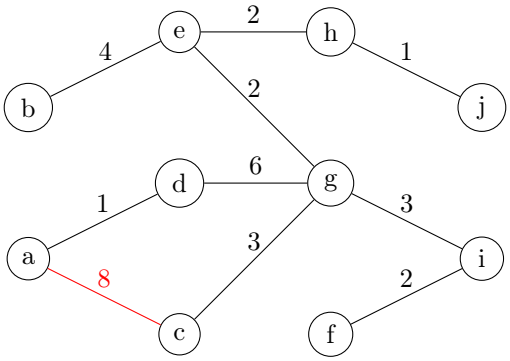
$g = 6$:



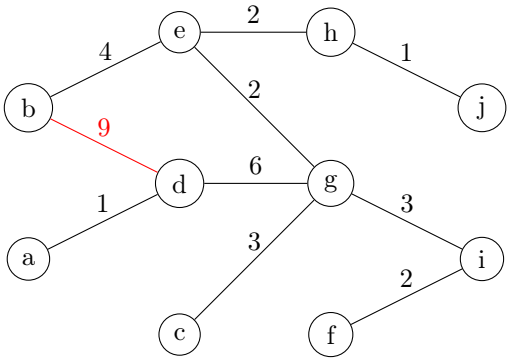
$g = 7$:



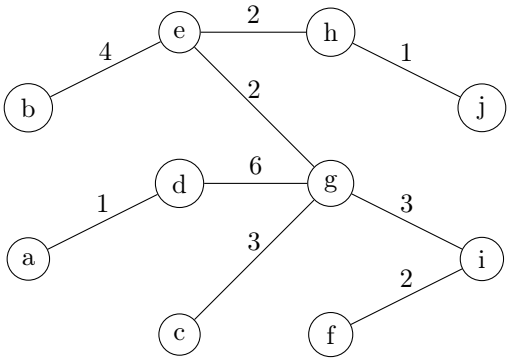
$g = 8$:



$g = 9$:



Finaler Baum:



Das Gewicht des Baums beträgt: $2 \cdot 1 + 3 \cdot 2 + 2 \cdot 3 + 1 \cdot 4 + 1 \cdot 6 = 24$

Aufgabe 6.2

Wir implementieren einen *Selection Sort* Algorithmus. Dieser wählt in n Iterationen jeweils aus dem Input-Array A das Minimum aus und fügt dieses dann der Reihe nach dem Output-Array *out* hinzu. Dabei wird dieses Element auch aus A entfernt, wodurch sich A stetig verkleinert, bis am Ende kein Element mehr übrig ist. Wir gehen davon aus, dass $\forall x \in A : x \in \mathbb{N}_0$.

```
1:  procedure SELECTIONSORT(A)
2:      n ← len(A)
3:      Initialize out[0, ..., n − 1] = −1
4:      for i ← 0 to n − 1 do
5:          idx ← 0
6:          val ← A[0]
7:          for j ← 0 to n − i do
8:              if A[j] < val then
9:                  val ← A[j]
10:                 idx ← j
11:         out[i] ← pop(A, idx)
12:     return out
```

Die Funktion „len(A)“ gibt die Länge des Arrays heraus. Die Funktion „pop(A, i)“ entfernt das i-te Elemente aus A und gibt es heraus (Die Länge von A wird dadurch um 1 kleiner).

Das Innere des zweiten for-loops wird stets $n + (n-1) + (n-2) + \ldots + 1 \leq n \cdot n \in \Theta(n^2)$ Mal ausgeführt, unabhängig von der Zusammensetzung des Input-Arrays.

Hierbei handelt es sich um einen Greedy-Algorithmus, der das Gesamtproblem in n Teilprobleme immer

kleinerer Größe unterteilt. Innerhalb jedes Teilproblems schreibt die Gewichtsfunktion $g(A_i) = A_i$ jedem Element sein Gewicht zu, was genau dem Wert dieses Elements entspricht. Die Anforderung lautet, das Gewicht zu minimieren, also stets den geringsten Wert der Teilmenge zu finden.

Aufgabe 6.3

In einer Adjazenzliste steht an jedem Eintrag u die Liste aller Zielknoten v. Folglich kann eine das Vorhandensein einer Kante (u, v) geprüft werden, indem die Liste von u nach v durchsucht wird. Im schlimmsten Fall ist diese Liste n lang. Bei der Adjazenzmatrix hingegen geschieht die Überprüfung in einem Aufruf, und zwar exakt an der Stelle (u, v).

Um alle von u ausgehenden Kanten aufzulisten, muss in der Adjazenzliste einfach die Liste bei u durchgegangen werden, was schlimmstenfalls n Schritte benötigt. In der Matrix muss einfach die gesamte Zeile durchgegangen werden und jedes Element mit einer 1 herausgegeben werden (das dauert exakt n Schritte).

Um hingegen alle zu v führenden Kanten aufzulisten, muss im Fall der Adjuzenzliste jedes Element u überprüft werden und darin nach dem Element v gesucht werden. Im schlimmsten Fall benötigt das bei $n \log n$ Kanten genau so viele Schritte. Bei der Adjuzenzmatrix hingegen kann analog zum vorigen Fall einfach die *Spalte* v ausgelesen werden und dabei jedes Element mit einer 1 herausgegeben werden. Erneut werden hier nur n Schritte benötigt.

Hat der Graph nicht $\mathcal{O}(n \log n)$ sondern $\mathcal{O}(n^2)$ viele Kanten, so ändert sich lediglich die maximale Suchzeit in der Adjuzenzliste im Fall c), da nun schlimmstenfalls n^2 viele Kanten durchsucht werden müssen.

Die Ergebnisse sind in der folgenden Tabelle zusammengefasst:

	Adjazenzliste	Adjazenzmatrix
a)	$\mathcal{O}(n)$	$\mathcal{O}(1)$
b)	$\mathcal{O}(n)$	$\mathcal{O}(n)$
c)	$\mathcal{O}(n \log n)$ ($\mathcal{O}(n^2)$ im Fall d)	$\mathcal{O}(n)$
Beispiel	u 0 → 1 / 1 → 2 / 2 → 2 → 3 / 3 → 0 /	v=2 u=2 0 1 0 0 0 0 1 0 0 0 1 1 1 0 0 0

Aufgabe 6.4

a) **Ausgangszustand:**

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	4	4	4	6	8	8	8	10	12	12	12	14	16	16	16

union(1,5):

$$\begin{aligned}\text{UFunion}(1,5) &\Rightarrow \text{UFunion}(\text{UFfind}(1), \text{UFfind}(5)) \\ &\Rightarrow \text{UFunion}(\text{UFfind}(2), \text{UFfind}(6)) \\ &\Rightarrow \text{UFunion}(\text{UFfind}(4), \text{UFfind}(8)) \\ &\Rightarrow \text{UFunion}(4,8)\end{aligned}$$

$$\Rightarrow \underbrace{A[1] = A[2] = 4}_{\text{UFfind(1)}}, \underbrace{A[5] = A[6] = 8}_{\text{UFfind(5)}}, \underbrace{A[4] = 8}_{\text{UFunion(4,8)}}$$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
4	**4**	4	**8**	**8**	**8**	8	8	10	12	12	12	14	16	16	16

union(11,13):

$$\begin{aligned}\text{UFunion}(11,13) &\Rightarrow \text{UFunion}(\text{UFfind}(11), \text{UFfind}(13)) \\ &\Rightarrow \text{UFunion}(\text{UFfind}(12), \text{UFfind}(14)) \\ &\Rightarrow \text{UFunion}(12, \text{UFfind}(16)) \\ &\Rightarrow \text{UFunion}(12,16)\end{aligned}$$

$$\Rightarrow \underbrace{A[11] = 12}_{\text{UFfind(11)}}, \underbrace{A[13] = A[14] = 16}_{\text{UFfind(13)}}, \underbrace{A[16] = 12}_{\text{UFunion(12,16)}}$$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
4	4	4	8	8	8	8	8	10	12	**12**	12	**14**	**16**	16	**12**

union(1,10):

$$\begin{aligned}\text{UFunion}(1,10) &\Rightarrow \text{UFunion}(\text{UFfind}(1), \text{UFfind}(10)) \\ &\Rightarrow \text{UFunion}(\text{UFfind}(4), \text{UFfind}(12)) \\ &\Rightarrow \text{UFunion}(\text{UFfind}(8), 12)) \\ &\Rightarrow \text{UFunion}(8,12)\end{aligned}$$

$$\Rightarrow \underbrace{A[1] = A[4] = 8}_{\text{UFfind(1)}}, \underbrace{A[10] = 12}_{\text{UFfind(10)}}, \underbrace{A[8] = 12}_{\text{UFunion(8,12)}}$$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
8	4	4	**8**	8	8	8	**12**	10	**12**	12	12	14	16	16	12

b) **find(2) ohne Pfadverkürzung**:

$$\text{UFfind}(2) \Rightarrow \text{UFfind}(4) \Rightarrow \text{UFfind}(8) \Rightarrow \text{UFfind}(12) = 12$$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
8	4	4	8	8	8	8	12	10	12	12	12	14	16	16	12

find(2) mit Pfadverkürzung:

$$\text{UFfind}(2) \Rightarrow \text{UFfind}(4) \Rightarrow \text{UFfind}(8) \Rightarrow \text{UFfind}(12) = 12$$
$$\Rightarrow A[2] = A[4] = A[8] = 12$$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
8	**12**	4	**12**	8	8	8	**12**	10	12	12	12	14	16	16	12

Aufgabe 6.5

a) Man kann die naheliegende Vermutung aufstellen, dass eine nicht-lineare Funktion w ein nicht-optimales bzw. ungleichmäßiges Ergebnis hervorruft. Wählen wir also zum Beispiel $w(i) = (n-i)^2$ und probieren $n = 8$ und betrachten welchen Gesamtwert Gauner 1 und Gauner 2 am Ende je erhalten würden:

i	$w(i)$	$G_1(i)$	$G_2(i)$
1	49	49	0
2	36	49	36
3	25	49	61
4	16	65	61
5	9	74	61
6	4	74	65
7	1	74	66
8	0	74	66

Tatsächlich weichen am Ende die Gesamtwerte um 8 voneinander ab, weshalb diese Funktion kein optimales Ergebnis liefert. Somit ist $w(i) = (n-i)^2$ eine mögliche Antwort auf die Aufgabenstellung.

b) Wählen wir stattdessen eine beliebige Funktion mit konstanter Ableitung, dann resultiert aus der Strategie stets eine optimale Lösung. Als Beispiel wählen wir $w(i) = 4(n-i)$ und erneut $n = 8$:

i	$w(i)$	$G_1(i)$	$G_2(i)$
1	28	28	0
2	24	28	24
3	20	28	44
4	16	44	44
5	12	56	44
6	8	56	52
7	4	56	56
8	0	56	56

Zumindest in diesem Beispiel ist das resultierende Ergebnis optimal - beide Gauner erhalten den selben Gesamtwert von 56. Dass dies für jeden konstanten Faktor und beliebige n der Fall ist, soll im Folgenden bewiesen werden:

Wir betrachten uns die finale Differenz der Gesamtwerte $G_1(n)$ und $G_2(n)$, die sich aus allen Einzelwerten wie folgt zusammensetzt. Als Funktion wählen wir die beliebige Funktion $w(i) = c \cdot (n-i)$ mit $c \in \mathbb{N}$:

$$
\begin{aligned}
G_1(n) - G_2(n) &\stackrel{!}{=} 0 \\
\Rightarrow 0 &= c\cdot(n-1) - c\cdot((n-2)+(n-3)) + \ldots + c\cdot(4+3) - c\cdot(2+1) + c\cdot 0 \qquad \mid \div c \\
&= \underbrace{(n-1)}_{\text{G1}} \underbrace{-((n-2)+(n-3))}_{\text{G2}} + \ldots \underbrace{+(4+3)}_{\text{G1}} \underbrace{-(2+1)}_{\text{G2}} + \underbrace{0}_{\text{G1}} \\
&= (n-1) + \sum_{i=2}^{n-1} \Big((-1)^{\lfloor i/2 \rfloor} (n-i) \Big) + 0 \\
&= (n-1) - \sum_{i=2}^{(n-4)/2} (2(n-i)-1) + \sum_{i=4}^{(n-2)/2} (2(n-i)-1) \\
&= (n-1) - \sum_{i=2}^{n/2-2} (2(n-i)-1) + \sum_{i=4}^{n/2-1} (2(n-i)-1) \\
&= (n-1) - \\
&- \left(\sum_{i=2}^{n/2} (2(n-i)-1) - (2(n-(n/2-1))-1) - (2(n-n/2)-1) \right) + \\
&+ \left(\sum_{i=2}^{n/2} (2(n-i)-1) - (2(n-2)-1) - (2(n-3)-1) - (2(n-n/2)-1) \right) \\
&= (n-1) + (2(n-(n/2-1))-1) - (2(n-2)-1) - (2(n-3)-1) \\
&= 2(6-n) \neq 0
\end{aligned}
$$