Diskrete und Geometrische Algorithmen

6. Übung am 07.12.2020

Richard Weiss Florian Schager Christian Sallinger Fabian Zehetgruber
Paul Winkler Christian Göth

Aufgabe 31. Ein binärer Suchbaum mit Blättern ist ein binärer Baum, sodass jeder Knoten entweder genau 2 oder keinen Nachfolger hat. Erstere Knoten werden *innere Knoten* genannt, letztere *Blätter* oder *externe Knoten*. Die *interne Pfadlänge* $I(T)$ eines Binärbaums T ist die Summe der Abstände (gemessen in Anzahl von Kanten) von der Wurzel zu allen internen Knoten, die *externe Pfadlänge* $E(T)$ ist die Summe der Abstände zu allen externen Knoten. Beweisen Sie:

a) Ein Binärbaum mit n internen Knoten hat $n + 1$ Blätter.

b) Zwischen interner und externer Pfadlänge besteht folgender Zusammenhang:

$$E(T) = I(T) + 2|T|,$$

wobei $|T|$ die Anzahl interner Knoten von T ist.

Lösung. So ein BSB ist entweder der 1-knotige (0-kantige) Baum oder besteht aus lauter verkehrt-rummen „v“-s.

a) IA($n = 0$):

Es gibt nur den Wurzel-Knoten. Dieser ist extern.

IS($n - 1 \mapsto n$):

Sei T ein BSB mit n internen Knoten. Lösche zwei Blätter, die ein *Kirschen-Paar* $\{v_1, v_2\}$ bilden, d.h. sodass wieder ein BSB T' entsteht. (Wenn es kein Kirschen-Paar gäbe, dann hätte ein Knoten von T genau einen Nachfolger!)

Der zugehörige T-interne Knoten, der die Kirschen verbindet v, wurde somit T'-extern. T' hat also einen internen Knoten (v) weniger als T, d.h. $n - 1$.

Laut IV hat T' daher $(n - 1) + 1 = n$ Blätter. Wenn wir das Kirschen-Paar $\{v_1, v_2\}$ wieder an T' drankleben (und die Kanten $\{v, v_1\}$ und $\{v, v_2\}$ ergänzen), kommt T raus. v ist nicht mehr Blatt (wir ziehen 1 ab) und die Kirschen kommen jeweils als Blatt dazu (wir addieren 2).

$$n - 1 + 2 = n + 1$$

Alternative: Die Anzahl der Kanten in unserem Baum ist genau $2n$ und daraus folgt bereits, dass unser Baum $2n + 1$ Knoten haben muss.

b) Sei T ein BSB.

IA($|T| = 0$):

Wieder trivial, alles 0.

IS$(n-1 \mapsto n)$:

Lösche wieder ein Kirschen-Paar $\{v_1, v_2\}$, wobei die Kirschen jeweils Tiefe m haben, und erhalte T'. Genauso wie vorher sieht man

$$|T'| = |T| - 1.$$

Der T'-externe bzw. T-interne Knoten v, der die Kirschen v_1 und v_2 verbindet hat Tiefe $m-1$.

$$\implies I(T') = I(T) - (m-1), \quad E(T') = E(T) - 2m + (m-1) = E(T) - (m+1)$$

Es ist Zeit, alles zusammenzustöpseln.

$$\begin{aligned} \implies E(T) = E(T') + (m+1) &\overset{\text{IV}}{=} I(T') + 2|T'| + m + 1 \\ &= I(T') + 2(|T| - 1) + m + 1 = I(T') + (m-1) + 2|T| = I(T) + 2|T| \end{aligned}$$

□

Aufgabe 32. Geben Sie, für jedes gerade $n \in \mathbb{N}$ paarweise unterschiedliche Schlüssel $x_1, \ldots, x_n$ an, so dass ein Suchbaum in Kammform (sh. Abbildung) entsteht wenn, beginnend mit dem leeren Suchbaum, die Schlüssel $x_1, \ldots, x_n$ in dieser Reihenfolge eingefügt werden.

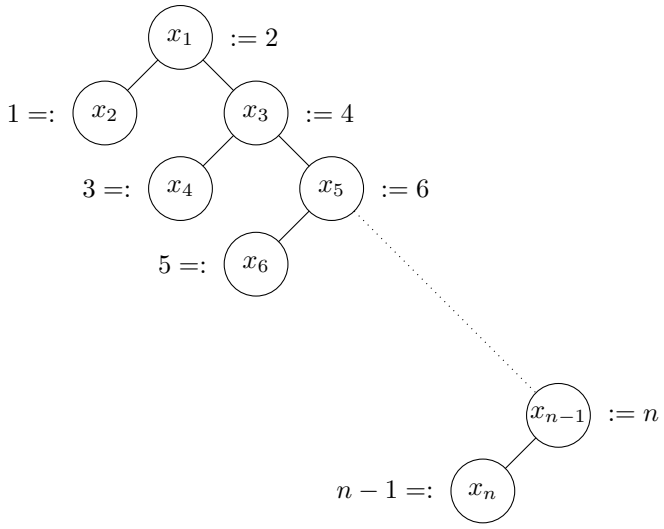
Lösung.

Definition 5.1. Ein Suchbaum ist ein Baum der die folgende *Suchbaumeigenschaft* hat: für alle Knoten v gilt:

1. Für jeden Knoten w im linken Teilbaum von v gilt: $w.D.x < v.D.x$.
2. Für jeden Knoten w im rechten Teilbaum von v gilt: $w.D.x > v.D.x$.

Algorithmus 14 Einfügen in einen Suchbaum

```
Prozedur EINFÜGEN(v, D)
    Falls v = NIL dann                          ▷ Leerer Suchbaum wird initialisiert
        Sei v_0 neuer Knoten
        v_0.D := D
        v_0.links := NIL
        v_0.rechts := NIL
        Antworte v_0
    sonst falls v.D.x = D.x dann
        Antworte "Schlüssel existiert bereits"
    sonst falls v.D.x < D.x dann
        v.rechts := EINFÜGEN(v.rechts, D)
        Antworte v
    sonst                                       ▷ v.D.x > D.x
        v.links := EINFÜGEN(v.links, D)
        Antworte v
    Ende Falls
Ende Prozedur
```



$$x_i := \begin{cases} i-1, & \text{falls } i \in 2\mathbb{N}, \\ i+1, & \text{falls } i \in 2\mathbb{N}+1, \end{cases} \qquad i = 1, \ldots, n$$

□

Aufgabe 33. Bestimmen Sie die minimale Anzahl von Knoten in einem balancierten Baum der Höhe h.

Lösung.

> Die *Höhe* $\mathrm{h}(v)$ eines Knotens v ist die maximale Anzahl von Knoten auf einem Pfad von v zu einem Blatt. Die Höhe des leeren Baums ist 0. Die Höhe eines nicht-leeren Baums ist die Höhe seiner Wurzel und damit ≥ 1.

> **Definition 5.2.** Sei T ein Suchbaum, v ein Knoten in T, T_l der linke Teilbaum von v und T_r der rechte Teilbaum von v. Dann ist der *Balancegrad* von v definiert als $\mathrm{bal}(v) = \mathrm{h}(T_\mathrm{r}) - \mathrm{h}(T_\mathrm{l})$. Ein Suchbaum T heißt *balanciert* falls für jeden Knoten v von T gilt: $|\mathrm{bal}(v)| \leq 1$.

Der balancierte Baum T_1 der Höhe $h = 1$ besteht nur aus der Wurzel, besitzt also nur $|E(T_1)| = 1$ Knoten. Der balancierte Baum T_2 der Höhe $h = 2$ besteht nur aus maximal 3 Knoten. Einen unteren Knote darf man entfernen und der Baum bleibt balanciert, d.h. $|E(T_2)| = 2$.
Für höhere Bäume T_h mit $h > 2$ können wir $|E(T_h)|$ bestimmen, indem wir rekursiv vorgehen: Der linke und rechte Teilbaum $(T_h)_\mathrm{l}$ bzw. $(T_h)_\mathrm{r}$ haben jeweils maximal Höhe $h-1$. Einer davon darf sogar Höhe $h-2$ haben, T_h bleibt balanciert und hat minimale Anzahl an Knoten.
Weil T_h balanciert ist, müssen auch die Teilbäume $(T_h)_\mathrm{l}$ und $(T_h)_\mathrm{r}$ balanciert sein. Weil T_h minimale Knoten-Zahl hat, müssen auch die Teilbäume $(T_h)_\mathrm{l}$ und $(T_h)_\mathrm{r}$ minimale Knoten-Zahl haben.

$$\Longrightarrow (T_h)_\mathrm{l} = T_{h-1}, \quad (T_h)_\mathrm{r} = T_{h-2}, \quad \text{oder} \quad (T_h)_\mathrm{l} = T_{h-2}, \quad (T_h)_\mathrm{r} = T_{h-1}$$

$$\Longrightarrow |E(T_h)| = |E((T_h)_\mathrm{l})| + |E((T_h)_\mathrm{r})| + 1 = |E(T_{h-1})| + |E(T_{h-2})| + 1$$

Wir haben also für die minimale Anzahl an Knoten die folgende Rekursionsgleichung.

$$m_h = m_{h-1} + m_{h-2} + 1, \quad m_1 = 1, \quad m_2 = 2$$

Wir haben es also mit einer inhomogenen Rekursionsgleichung 2-ter Ordnung zu tun.

Satz 4.3. *Sei K ein Körper, seien $c_0, \ldots, c_{k-1}, d \in K$, dann hat (4.2) die Lösung $(x_n)_{n\geq 0}$ definiert durch $x_n = y_n - \frac{d}{C-1}$ wobei $C = \sum_{i=0}^{k-1} c_i$ und y_n ist Lösung von*

$$y_n = x_n + \frac{d}{C-1} \text{ für } 0 \leq n < k$$
$$y_n = c_{k-1}y_{k_1} + \cdots + c_0 y_{n-k} \text{ für } n \geq k$$

Zuerst lösen wir die zugehörige homogene Rekursionsgleichung.

$$n_h = n_{h-1} + n_{h-2}, \quad n_1 = 2, \quad n_2 = 3$$

Das ist genau die, um 1 im Index verschobene, Fibonacci-Folge.

$$\implies n_h = F_{h+1} = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^{h+1} - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^{h+1}$$

Nun wieder zur inhomogenen Rekursionsgleichung. Diese hat schließlich die folgende Lösung.

$$\implies m_h = n_h - 1 = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^{h+1} - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^{h+1} - 1$$

□

Aufgabe 34. Gegeben ist der gerichtete Graph $G = (V, E)$ mit $V = \{a, b, \ldots, m\}$ und

$$E = \{ag, ba, be, cb, cd, dj, ec, ef, fd, fh, ge, hg, hi, hk, if, il, ji, jm, ka, lg, ml\}.$$

Erstellen Sie die Liste der besuchten Knoten, die der Reihenfolge während einer Breitensuche bzw. während einer Tiefensuche entspricht.

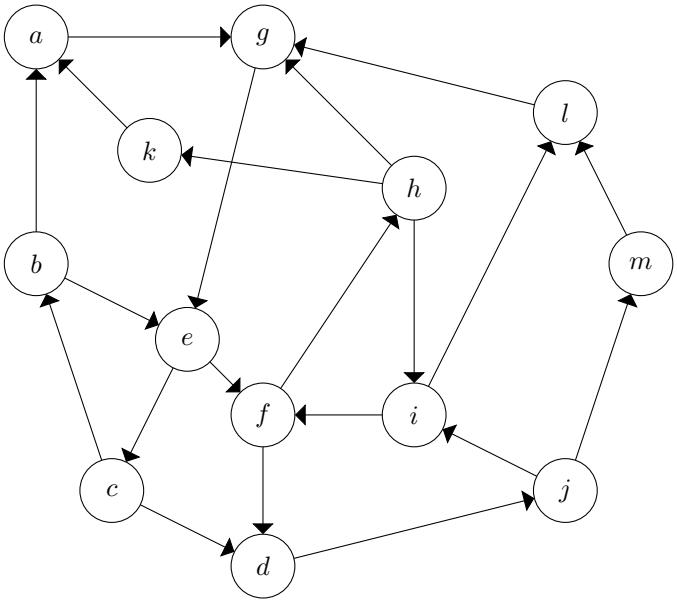
Lösung.

Algorithmus 24 Breitensuche (engl. *breadth-first search*)

```
Prozedur BREITENSUCHE(V, E, u)
    Sei bekannt neues Datenfeld der Länge |V|, überall mit falsch initialisiert
    Sei Q eine neue Warteschlange, leer initialisiert
    bekannt[u] := wahr
    HINZUFÜGEN(Q, u)
    Solange Q nicht leer
        v := ENTFERNEN(Q)
        P(v)
        Für jede Kante (v, w) ∈ E
            Falls bekannt[w] = falsch dann
                bekannt[w] = wahr
                HINZUFÜGEN(Q, w)
            Ende Falls
        Ende Für
    Ende Solange
Ende Prozedur
```

Wir starten die Breitensuche im Knoten a.

$0:$	$Q = [a]$
$1: [a],$	$Q = [g]$
$2: [a, g],$	$Q = [e]$
$3: [a, g, e],$	$Q = [c, f]$
$4: [a, g, e, c],$	$Q = [f, b, d]$
$5: [a, g, e, c, f],$	$Q = [b, d, h]$
$6: [a, g, e, c, f, b],$	$Q = [d, h]$
$7: [a, g, e, c, f, b, d],$	$Q = [h, j]$
$8: [a, g, e, c, f, b, d, h],$	$Q = [j, i, k]$
$9: [a, g, e, c, f, b, d, h, j],$	$Q = [i, k, m]$
$10: [a, g, e, c, f, b, d, h, j, i],$	$Q = [k, m, l]$
$11: [a, g, e, c, f, b, d, h, j, i, k],$	$Q = [m, l]$
$12: [a, g, e, c, f, b, d, h, j, i, k, m],$	$Q = [l]$
$13: [a, g, e, c, f, b, d, h, j, i, k, m, l]$	

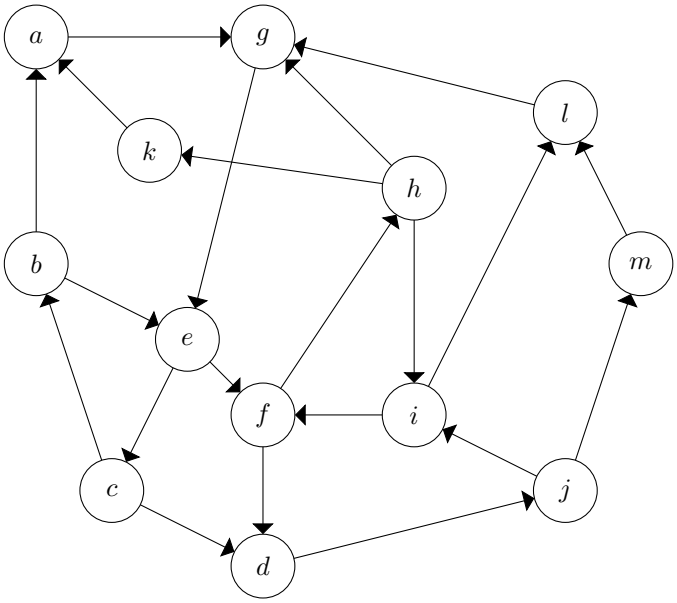


Algorithmus 25 Tiefensuche (engl. *depth-first search*)

```
Prozedur TIEFENSUCHE(V, E, u)
    Sei bekannt neues Datenfeld der Länge |V|, überall mit falsch initialisiert
    Sei S ein neuer Stapel, leer initialisiert
    bekannt[u] := wahr
    HINZUFÜGEN(S, u)
    Solange S nicht leer
        v := ENTFERNEN(S)
        P(v)
        Für jede Kante (v, w) ∈ E
            Falls bekannt[w] = falsch dann
                bekannt[w] = wahr
                HINZUFÜGEN(S, w)
            Ende Falls
        Ende Für
    Ende Solange
Ende Prozedur
```

Wir starten die Tiefensuche im Knoten a.

$0:$	$S = [a]$
$1: [a],$	$S = [g]$
$2: [a, g],$	$S = [e]$
$3: [a, g, e],$	$S = [c, f]$
$4: [a, g, e, f],$	$S = [c, d, h]$
$5: [a, g, e, f, h],$	$S = [c, d, i, k]$
$6: [a, g, e, f, h, k],$	$S = [c, d, i]$
$7: [a, g, e, f, h, k, i],$	$S = [c, d, l]$
$8: [a, g, e, f, h, k, i, l],$	$S = [c, d]$
$9: [a, g, e, f, h, k, i, l, d],$	$S = [c, j]$
$10: [a, g, e, f, h, k, i, l, d, j],$	$S = [c, m]$
$11: [a, g, e, f, h, k, i, l, d, j, m],$	$S = [c]$
$12: [a, g, e, f, h, k, i, l, d, j, m, c],$	$S = [b]$
$13: [a, g, e, f, h, k, i, l, d, j, m, c, b]$	



□

Aufgabe 35. Der Durchmesser eines Graphen ist die maximale Distanz zwischen zwei Knoten. Entwerden Sie einen effizienten Algorithmus zur Bestimmung des Durchmessers eines Baumes und analysieren Sie dessen Laufzeit.

Lösung. Der Algorithmus ist so konstruiert, dass für jeden Knoten $v \in V$ der längste Pfad durch diesen Knoten bestimmt wird und davon dann das max genommen wird. Diesen längsten Pfad bestimmen wir, indem wir den Knoten v als Wurzel setzen und die Höhen der zwei höchsten Teilbäume, die nach wegstreichen der Wurzel entstehen, addieren. Die Höhen werden dabei rekursiv genauso bestimmt, nur ohne die Addition. Das funktioniert deswegen, weil die Höhe eines Teilbaums die Anzahl der Knoten im maximal langen Pfad von einem Blatt zur Teilbaum-Wurzel ist. Das ist aber genau die Anzahl der Kanten vom Blatt zur echten Wurzel.

Algorithm 1 Durchmesser eines Baumes

```
1: procedure DurchmesserBaum(B)
2:     V, E := B
3:     Sei L neues Datenfeld der Länge |V|
4:     for v ∈ V do
5:         Z := Zusammenhangskomponenten(B − v)
6:         if Z.Länge = 0 then ⟹ B − v ist leer
7:             L[v] := 0
8:         else if Z.Länge = 1 then
9:             L[v] := Höhe(B − v) + 1
10:        else if Z.Länge ≥ 2 then
11:            H := [Höhe(B′, v′) : B′ ∈ Z, {v, v′} ∈ E]
12:            Seien h_1, h_2 ∈ H die beiden (verschiedenen) größten Einträge.
13:            L[v] := h_1 + h_2
14:        end if
15:    end for
16:    return max L
17: end procedure
```

Algorithm 2 Höhe eines Baumes

```
1: procedure Höhe(B, v)
2:     V, E := B
3:     if |V| = 1 then
4:         return 1
5:     else if |V| ≥ 2 then
6:         Z := Zusammenhangskomponenten(B − v)
7:         return max [Höhe(B′, v′) : B′ ∈ Z, {v, v′} ∈ E] + 1
8:     end if
9: end procedure
```

Aufwand-Bestimmung:
In Zeile 4 sehen wir, dass der Aufwand wohl $\mathcal{O}(|V| \cdot A)$ sein wird. A bezeichnet den Aufwand innerhalb der Schleife. Die Bestimmung der Zusammenhangskomponenten in Zeile 5 ist in unserem Fall einfach und trägt nicht groß zum Aufwand bei.
Die Zahl *Z.Länge* der Zusammenhangskomponenten (bzw. Teilbäume) und deren Höhen verhalten sich invers zueinander. Wenn es mehr Zusammenhangskomponenten gibt, dann sind diese jeweils nicht so hoch und umgekehrt.

$$\sum_{B \in Z} \text{Höhe}(B, \cdot) \leq \sum_{(V', E') \in Z} |V'| + 1 = |V|$$

Bei der Bestimmung der Höhe einer Zusammenhangskomponente wird nun aber jeder Knoten innerhalb dieser Zusammenhangskomponente besucht. Die Höhe von jeder Zusammenhangskomponente wird bestimmt. Insgesamt besuchen wir also alle Knoten (bis auf die weggestrichene Wurzel v). Wir haben somit Aufwand $A = \mathcal{O}(|V|)$ innerhalb der Schleife.
Insgesamt Insgesamt hat der Algorithmus somit Aufwand $\mathcal{O}(|V|^2)$.

□

Aufgabe 36.

Modifizieren Sie die Prozeduren für die Tiefensuche so, dass bei Eingabe eines ungerichteten Graphen $V = (V, E)$ für jeden Knoten v ein weiteres Attribut $K(v)$ bestimmt wird, das die folgenden beiden Bedingungen erfüllt:

(a) $K(v) \in \{1, 2, \ldots, k\}$, wobei k die Anzahl der Zusammenhangskomponenten ist.

(b) $K(u) = K(v)$ genau dann, wenn u und v in der selben Zusammenhangskomponente liegen.

Lösung. Siehe Algroithmus

Algorithm 3 Einteilung in Zusammenhangskomponenten

```
1:  procedure ZUSAMMENHANGSKOMPONENTEN(V, E)
2:      Sei bekannt neues Datenfeld der Länge |V|, überall mit falsch initialisiert.
3:      Sei K neues Datenfeld der Länge |V|
4:      Sei S ein neuer Stapel, leer initialisiert
5:      l := 0
6:      n := |V|
7:      while n > 0 do
8:          l := l + 1
9:          Wähle Knoten u mit bekannt[u] = falsch
10:         bekannt[u] := wahr
11:         HINZUFÜGEN(S, u)
12:         while S nicht leer do
13:             v := ENTFERNEN(S)
14:             K[v] := l
15:             n := n − 1
16:             for jede Kante (v, w) ∈ E do
17:                 if bekannt[w] = falsch then
18:                     bekannt[w] := wahr
19:                     HINZUFÜGEN(S, w)
20:                 end if
21:             end for
22:         end while
23:     end while
24: end procedure
```

□

Diskrete und Geometrische Algorithmen

7. Übung am 14.12.2020

Richard Weiss Florian Schager Christian Sallinger Fabian Zehetgruber
Paul Winkler Christian Göth

Aufgabe 37. Geben Sie einen Algorithmus mit Laufzeit $\mathcal{O}(|V|+|E|)$ an, dessen Eingabe ein gerichteter azyklischer Graph $G = (V, E)$ und zwei Knoten s und t sind, der die Anzahl der (gerichteten) Wege von s nach t berechnet. Beispielsweise enthält der Graph aus Aufgabe 50 drei Wege von b nach i nämlich $b-d-e-h-i$, $b-d-f-h-i$ und $b-g-i$. (Anmerkung: der Algorithmus soll die Pfade bloß zählen, nicht ausgeben)

Hinweis: Topologisches Sortieren

Lösung.

Algorithm 1 Anzahl der gerichteten Wege eines gerichteten azyklischen Graphen $G = (V, E)$ von s nach t

```
1: procedure AnzahlPfade(E, s, t)
2:     Setze v.n := +∞ für alle v ∈ V
3:     return AnzahlPfadeRek(E, s, t)
4: end procedure
5:
6: procedure AnzahlPfadeRek(E, s, t)
7:     if s.n = +∞ then
8:         if s = t then
9:             s.n := 1
10:        else
11:            s.n := 0
12:            for (s, w) ∈ E do
13:                s.n := s.n + AnzahlPfadeRek(E, w, t)
14:            end for
15:        end if
16:    end if
17:    return v.n
18: end procedure
```

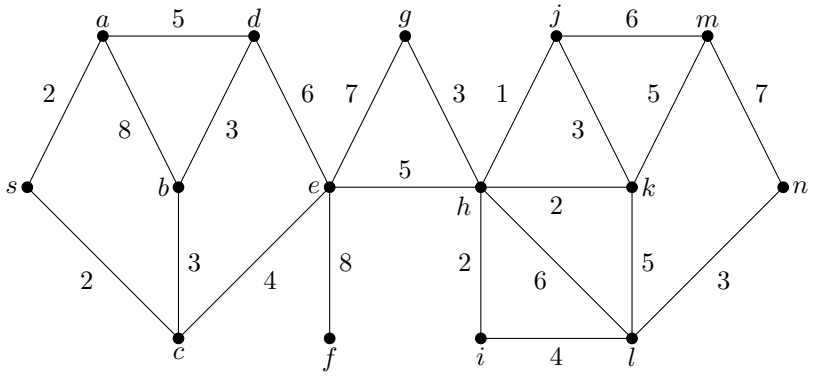
□

Lösung. Bestimme zuerst mittels Tiefen-/Breitensuche den Teilgraphen $G = (V', E')$, welcher aus all jenen Knoten besteht, welche von s ausgehend erreicht werden können mit $E' = E \cap V' \times V'$. Dann können wir folgenden, ans topologische Sortieren angelehnten Algorithmus anwenden: □

Algorithm 2 Anzahl der gerichteten Wege eines gerichteten azyklischen Graphen $G = (V, E)$ von s nach t

```
1: procedure AnzahlGerichteteWege(V, E, s, t)
2:     Sei pfadeZu ein neues Datenfeld der Länge |V| überall mit 0 initialisiert
3:     Sei eingangsgrad ein neues Datenfeld der Länge |V| überall mit 0 initialisiert
4:     for alle (v, w) ∈ E do
5:         Eingangsgrad[w] := Eingangsgrad[w] + 1
6:     end for
7:     Sei M neue Warteschlange oder Stapel leer initialisiert
8:     Hinzufügen(M, s)
9:     pfadeZu[s] := 1
10:    while M nicht leer do
11:        v := Entfernen(M)
12:        for alle (v, w) ∈ E do
13:            pfadeZu[w] := pfadeZu[w] + pfadeZu[v]
14:            eingangsgrad[w] := eingangsgrad[w] − 1
15:            if eingangsgrad[w] = 0 then
16:                Hinfufügen(M, w)
17:            end if
18:        end for
19:    end while
20:    return pfadeZu[t]
21: end procedure
```

Aufgabe 38. Erklären Sie die Funktionsweise der Algorithmen von Kruskal und Prim anhand der Konstruktion eines maximalen Spannbaums (eines spannenden Baums mit maximalen Gewicht) mittels Kruskal's Algorithmus und eines minimalen Spannbaums mittels Prim's Algorithmus für den Wurzelknoten s im folgenden kantenbewerteten Graph:



Lösung.

1. Algorithmus (von Kruskal):

   Der Algorithmus von Kruskal erhält als Eingabe einen zusammenhängenden endlichen (möglicherweise sogar gerichteten) Graphen $G = (V, E)$ und eine Kostenfunktion $c : E \to \mathbb{R}^+$.

$$G = (V, E), \quad V = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, s\},$$

$$E = \{\{a,b\},\{a,d\},\{a,s\},\{b,c\},\{b,d\},\{c,e\},\{c,s\},\{d,e\},\{e,f\},\{e,g\},\{e,h\},\\ \{g,h\},\{h,i\},\{h,j\},\{h,k\},\{h,l\},\{i,l\},\{j,k\},\{j,m\},\{k,l\},\{k,m\},\{l,n\},\{m,n\}\}$$

$$\begin{array}{llll} c(\{a,b\}) = 8, & c(\{a,d\}) = 5, & c(\{a,s\}) = 2, & \\ c(\{b,c\}) = 3, & c(\{b,d\}) = 3, & & \\ c(\{c,e\}) = 4, & c(\{c,s\}) = 2, & & \\ c(\{d,e\}) = 6, & & & \\ c(\{e,f\}) = 8, & c(\{e,g\}) = 7, & c(\{e,h\}) = 5, & \\ c(\{g,h\}) = 3, & & & \\ c(\{h,i\}) = 2, & c(\{h,j\}) = 1, & c(\{h,k\}) = 2, & c(\{h,l\}) = 6, \\ c(\{i,l\}) = 4, & & & \\ c(\{j,k\}) = 3, & c(\{j,m\}) = 6, & & \\ c(\{k,l\}) = 5, & c(\{k,m\}) = 5, & & \\ c(\{l,n\}) = 3, & & & \\ c(\{m,n\}) = 7 & & & \end{array}$$

B wird die Kanten-Menge des gewünschten minimalen Spannbaums von G. Diese startet mit der leeren Menge $\emptyset$ und es werden sukzessive passende Kanten hinzugefügt.

Zuerst wird allerdings der Graph in einen Wald P aus 1-punktigen Bäumen partitioniert.

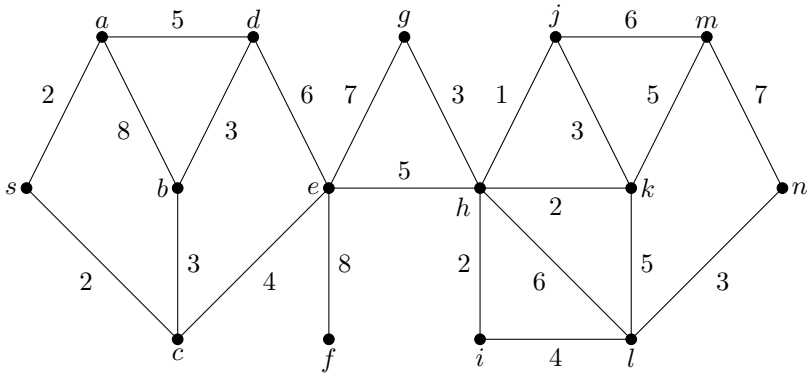
$$P_0 := \{(\{a\},\emptyset),(\{b\},\emptyset),(\{c\},\emptyset),(\{d\},\emptyset),(\{e\},\emptyset),(\{f\},\emptyset),(\{g\},\emptyset),\\ (\{h\},\emptyset),(\{i\},\emptyset),(\{j\},\emptyset),(\{k\},\emptyset),(\{l\},\emptyset),(\{m\},\emptyset),(\{n\},\emptyset),(\{s\},\emptyset)\}$$

Diese Bäume werden mit passenden Kanten $\in E$ zu einem minimalen Spannbaum verknüpft. Dazu iteriert man über das aus E bzgl. c von oben nach unten sortierte Datenfeld.

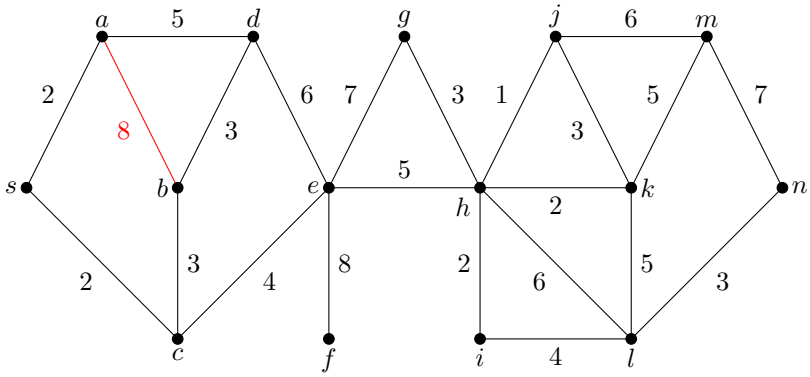
$$\begin{array}{llllllllllll} c(\{a,b\}) = 8 & \geq & c(\{e,f\}) = 8 & \geq & & & & & & & & \\ c(\{e,g\}) = 7 & \geq & c(\{m,n\}) = 7 & \geq & & & & & & & & \\ c(\{d,e\}) = 6 & \geq & c(\{h,l\}) = 6 & \geq & c(\{j,m\}) = 6 & \geq & & & & & & \\ c(\{a,d\}) = 5 & \geq & c(\{e,h\}) = 5 & \geq & c(\{k,l\}) = 5 & \geq & c(\{k,m\}) = 5 & \geq & & & & \\ c(\{c,e\}) = 4 & \geq & c(\{i,l\}) = 4 & \geq & & & & & & & & \\ c(\{b,c\}) = 3 & \geq & c(\{b,d\}) = 3 & \geq & c(\{g,h\}) = 3 & \geq & c(\{j,k\}) = 3 & \geq & c(\{l,n\}) = 3 & \geq & & \\ c(\{a,s\}) = 2 & \geq & c(\{c,s\}) = 2 & \geq & c(\{h,i\}) = 2 & \geq & c(\{h,k\}) = 2 & \geq & & & & \\ c(\{h,j\}) = 1 & & & & & & & & & & & \end{array}$$

$$E \mapsto [\{a,b\},\{e,f\},\{e,g\},\{m,n\},\{d,e\},\{h,l\},\{j,m\},\{a,d\},\{e,h\},\{k,l\},\{k,m\},\\ \{c,e\},\{i,l\},\{b,c\},\{b,d\},\{g,h\},\{j,k\},\{l,n\},\{a,s\},\{c,s\},\{h,i\},\{h,k\},\{h,j\}]$$

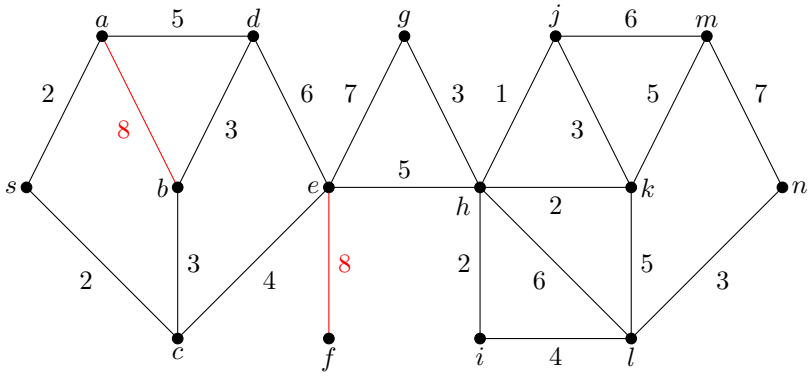
Es werden also am ehesten jene Kanten als Verknüpfung verwendet, die maximale Kosten liefern. Wir führen exemplarisch ein paar Schritte durch, sodass alle (2) Fälle abgedeckt sind.



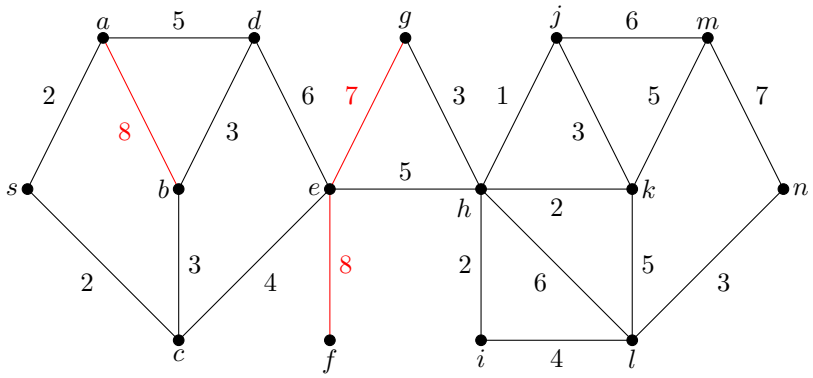
$$P_0 \overset{\{a,b\}}{\mapsto} P_1 := \{(\{a,b\},\{\{a,b\}\}),(\{c\},\emptyset),(\{d\},\emptyset),(\{e\},\emptyset),(\{f\},\emptyset),(\{g\},\emptyset),$$
$$(\{h\},\emptyset),(\{i\},\emptyset),(\{j\},\emptyset),(\{k\},\emptyset),(\{l\},\emptyset),(\{m\},\emptyset),(\{n\},\emptyset),(\{s\},\emptyset)\}$$



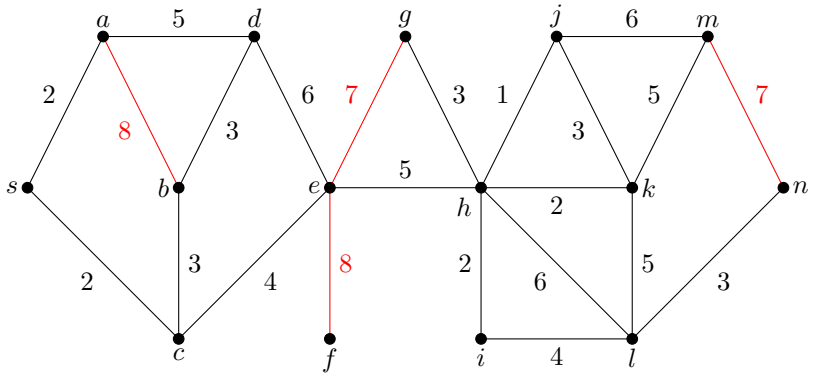
$$P_1 \overset{\{e,f\}}{\mapsto} P_2 := \{(\{a,b\},\{\{a,b\}\}),(\{c\},\emptyset),(\{d\},\emptyset),(\{e,f\},\{\{e,f\}\}),(\{g\},\emptyset),$$
$$(\{h\},\emptyset),(\{i\},\emptyset),(\{j\},\emptyset),(\{k\},\emptyset),(\{l\},\emptyset),(\{m\},\emptyset),(\{n\},\emptyset),(\{s\},\emptyset)\}$$



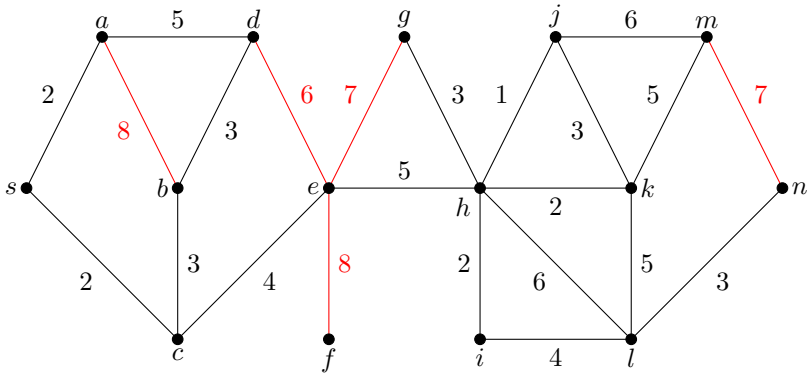
$$P_2 \overset{\{e,g\}}{\mapsto} P_3 := \{(\{a,b\},\{\{a,b\}\}),(\{c\},\emptyset),(\{d\},\emptyset),(\{e,f,g\},\{\{e,f\},\{e,g\}\}),$$
$$(\{h\},\emptyset),(\{i\},\emptyset),(\{j\},\emptyset),(\{k\},\emptyset),(\{l\},\emptyset),(\{m\},\emptyset),(\{n\},\emptyset),(\{s\},\emptyset)\}$$



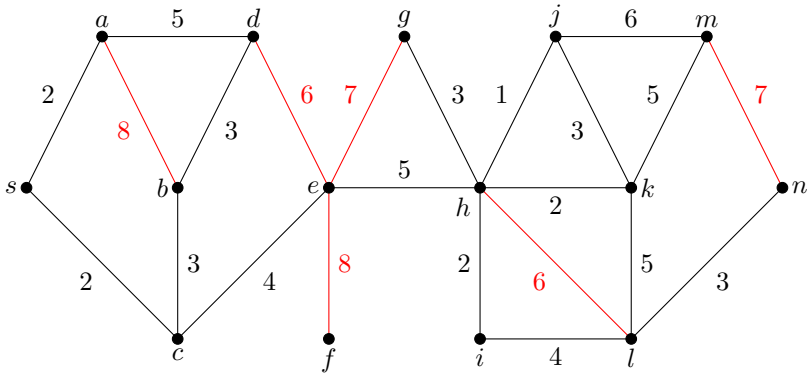
$$P_3 \overset{\{m,n\}}{\mapsto} P_4 := \{(\{a,b\},\{\{a,b\}\}),(\{c\},\emptyset),(\{d\},\emptyset),(\{e,f,g\},\{\{e,f\},\{e,g\}\}),$$
$$(\{h\},\emptyset),(\{i\},\emptyset),(\{j\},\emptyset),(\{k\},\emptyset),(\{l\},\emptyset),(\{m,n\},\{\{m,n\}\}),(\{s\},\emptyset)\}$$



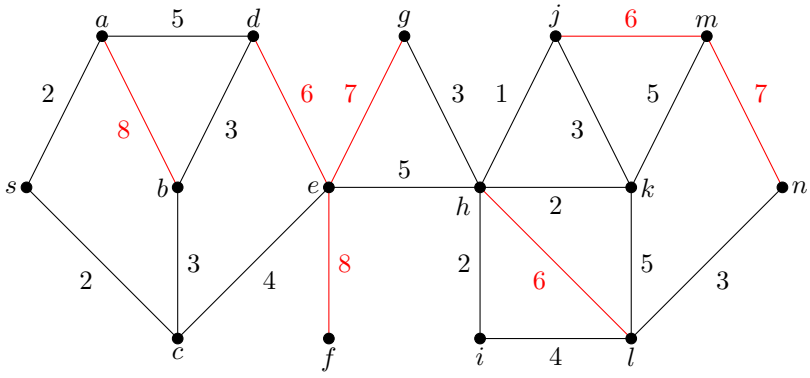
$$P_4 \overset{\{d,e\}}{\mapsto} P_5 := \{(\{a,b\},\{\{a,b\}\}),(\{c\},\emptyset),(\{d,e,f,g\},\{\{d,e\},\{e,f\},\{e,g\}\}),$$
$$(\{h\},\emptyset),(\{i\},\emptyset),(\{j\},\emptyset),(\{k\},\emptyset),(\{l\},\emptyset),(\{m,n\},\{\{m,n\}\}),(\{s\},\emptyset)\}$$



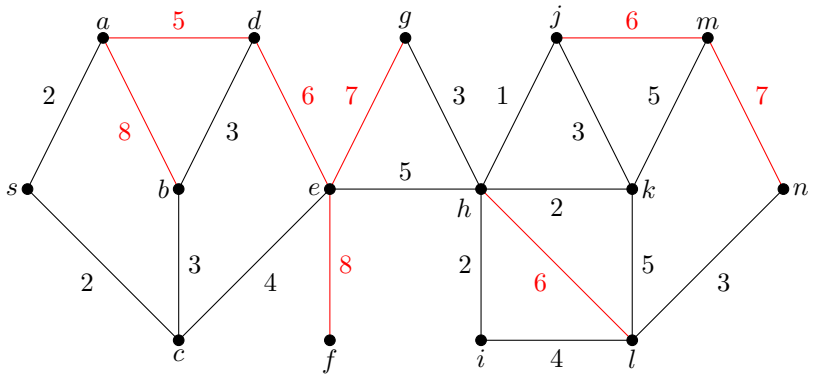
$P_5 \overset{\{h,l\}}{\mapsto} P_6 := \{(\{a,b\},\{\{a,b\}\}),(\{c\},\emptyset),(\{d,e,f,g\},\{\{d,e\},\{e,f\},\{e,g\}\}),$
$(\{h,l\},\{\{h,l\}\}),(\{i\},\emptyset),(\{j\},\emptyset),(\{k\},\emptyset),(\{m,n\},\{\{m,n\}\}),(\{s\},\emptyset)\}$



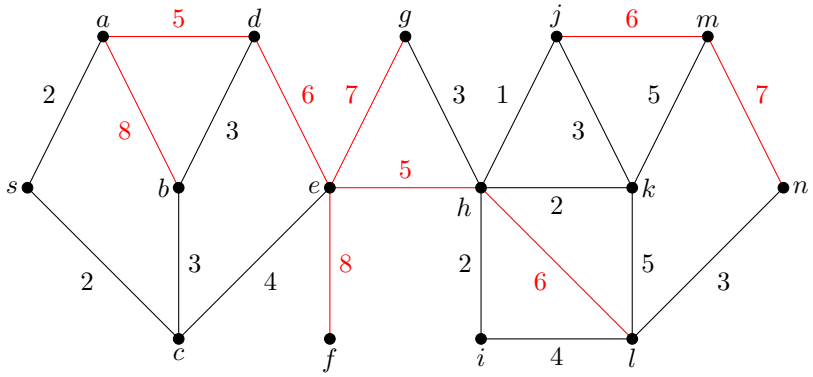
$P_6 \overset{\{j,m\}}{\mapsto} P_7 := \{(\{a,b\},\{\{a,b\}\}),(\{c\},\emptyset),(\{d,e,f,g\},\{\{d,e\},\{e,f\},\{e,g\}\}),$
$(\{h,l\},\{\{h,l\}\}),(\{i\},\emptyset),(\{j,m,n\},\{\{j,m\},\{m,n\}\}),(\{k\},\emptyset),(\{s\},\emptyset)\}$



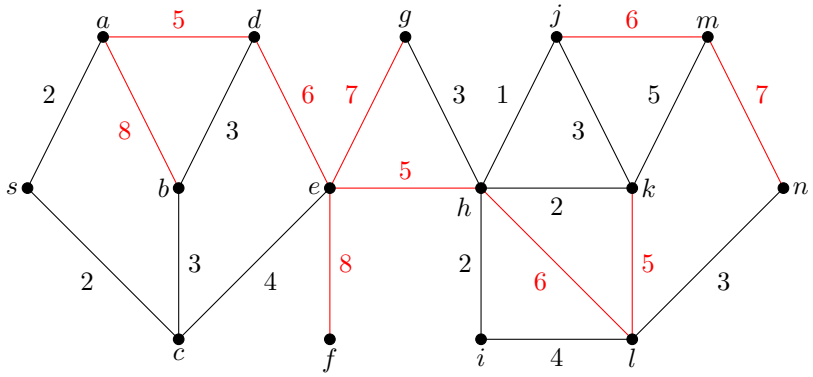
$$P_7 \overset{\{a,d\}}{\mapsto} P_8 := \{(\{a,b,d,e,f,g\},\{\{a,b\},\{d,e\},\{e,f\},\{e,g\}\}),(\{c\},\emptyset),$$
$$(\{h,l\},\{\{h,l\}\}),(\{i\},\emptyset),(\{j,m,n\},\{\{j,m\},\{m,n\}\}),(\{k\},\emptyset),(\{s\},\emptyset)\}$$



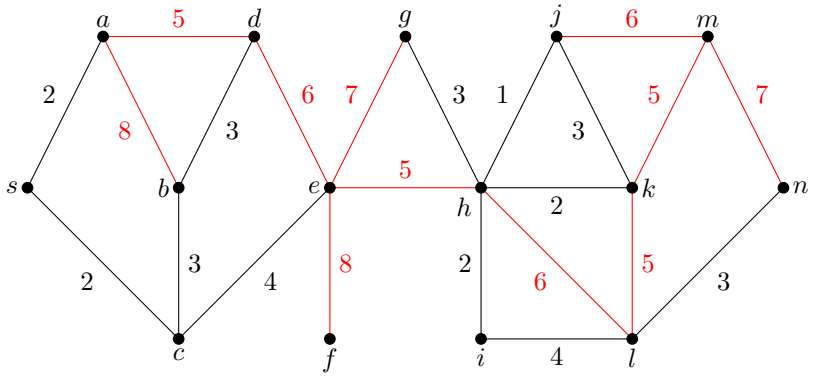
$$P_8 \overset{\{e,h\}}{\mapsto} P_9 := \{(\{a,b,d,e,f,g,h,l\},\{\{a,b\},\{d,e\},\{e,f\},\{e,g\},\{h,l\}\}),(\{c\},\emptyset),$$
$$(\{i\},\emptyset),(\{j,m,n\},\{\{j,m\},\{m,n\}\}),(\{k\},\emptyset),(\{s\},\emptyset)\}$$



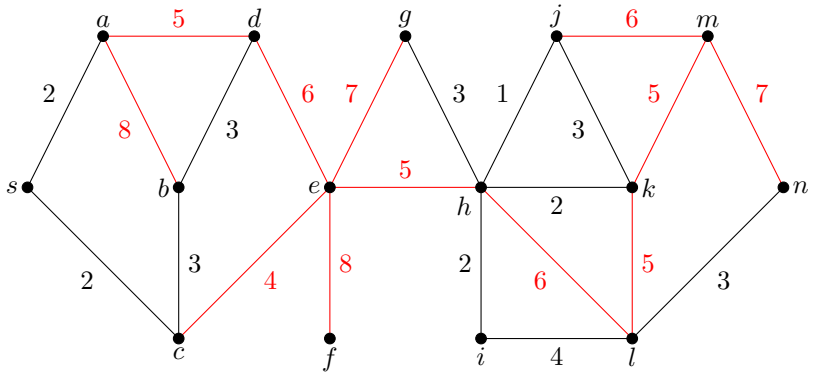
$$P_9 \overset{\{k,l\}}{\mapsto} P_{10} := \{(\{a,b,d,e,f,g,h,k,l\},\{\{a,b\},\{d,e\},\{e,f\},\{e,g\},\{h,l\},\{k,l\}\}),(\{c\},\emptyset),$$
$$(\{i\},\emptyset),(\{j,m,n\},\{\{j,m\},\{m,n\}\}),(\{s\},\emptyset)\}$$



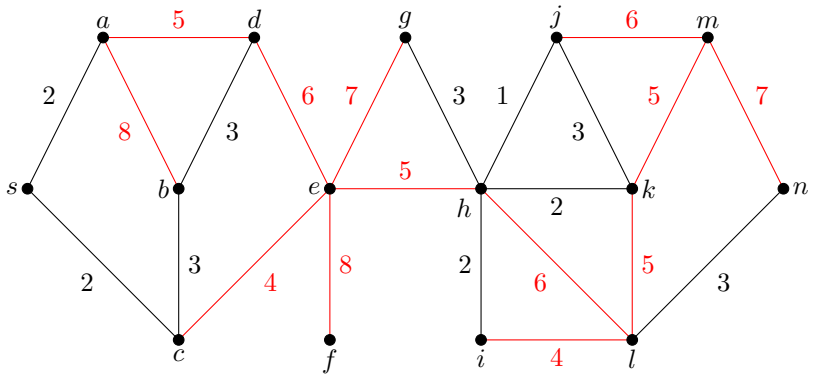
$P_{10} \overset{\{k,m\}}{\mapsto} P_{11} := \{(\{a,b,d,e,f,g,h,j,k,l,m,n\},$
$\{\{a,b\},\{d,e\},\{e,f\},\{e,g\},\{h,l\},\{j,m\},\{k,l\},\{m,n\}\}),(\{c\},\emptyset),(\{i\},\emptyset),(\{s\},\emptyset)\}$



$P_{11} \overset{\{c,e\}}{\mapsto} P_{12} := \{(\{a,b,c,d,e,f,g,h,j,k,l,m,n\},$
$\{\{a,b\},\{c,e\},\{d,e\},\{e,f\},\{e,g\},\{h,l\},\{j,m\},\{k,l\},\{m,n\}\}),(\{i\},\emptyset),(\{s\},\emptyset)\}$

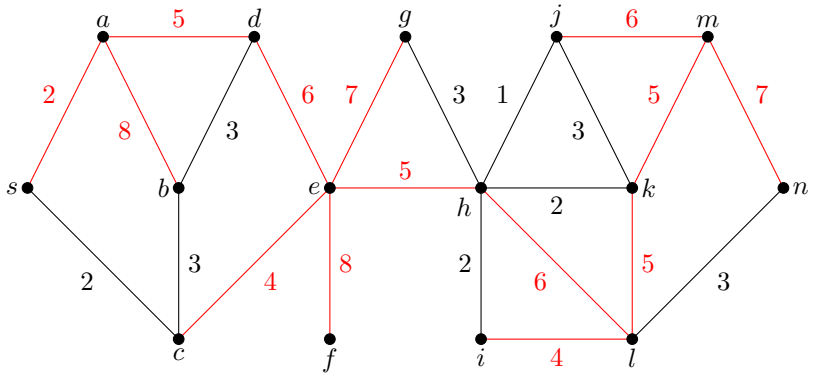


$$P_{12} \overset{\{i,l\}}{\mapsto} P_{13} := \{(\{a,b,c,d,e,f,g,h,i,j,k,l,m,n\},$$
$$\{\{a,b\},\{c,e\},\{d,e\},\{e,f\},\{e,g\},\{h,l\},\{i,l\}\{j,m\},\{k,l\},\{m,n\}\}),(\{s\},\emptyset)\}$$



$$P_{13} \overset{\{b,c\}}{\mapsto} P_{14} := P_{13}$$

...



Nachdem G als zusammenhängend vorausgesetzt war, werden alle ursprünglichen Äquivalenzklassen (d.h. 1-punktige Bäume) durch eine Kante getroffen.

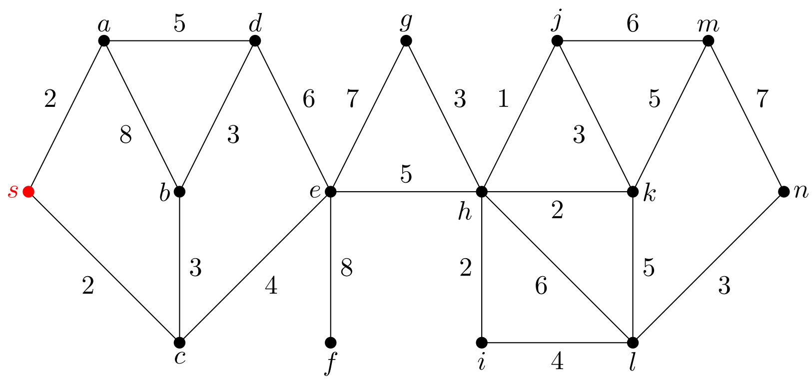
2. Algorithmus (von Prim):

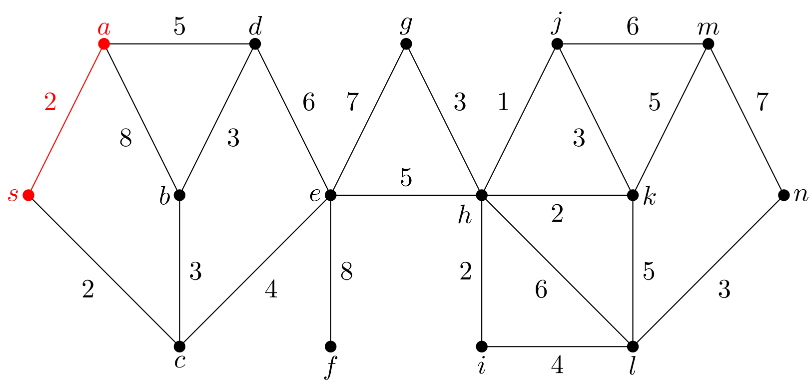
Definition 7.1. Sei $G = (V, E)$ und $c : E \to \mathbb{R}_{\geq 0}$, sei $G' = (V', E')$ ein Teilgraph von G und sei $v \in V \setminus V'$. Dann sind die *Anschlusskosten* von v an G' bezüglich c

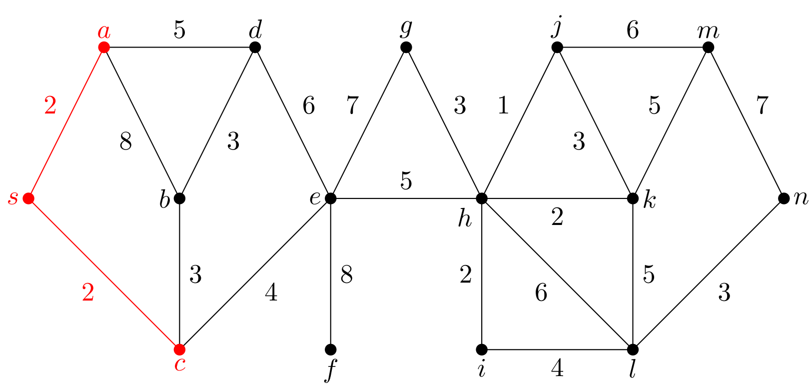
$$\min\{c(\{u,v\}) \mid \{u,v\} \in E, u \in V'\}$$

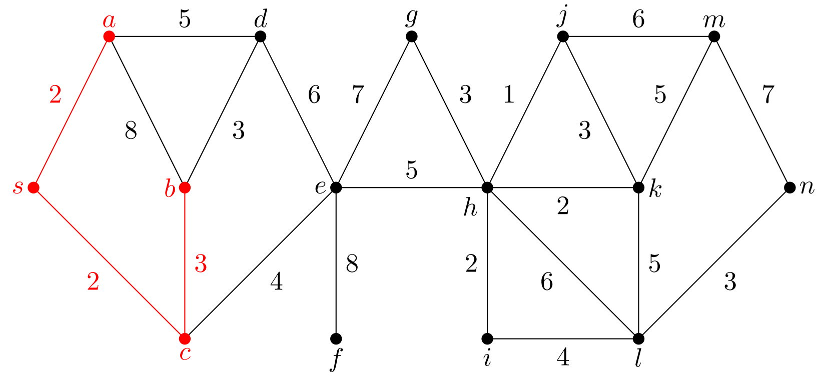
bzw. $+\infty$ falls diese Menge leer ist.

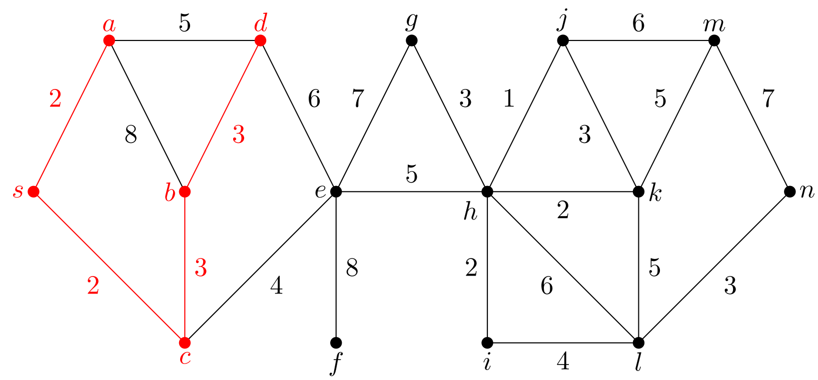
Der Algorithmus von Prim geht wie folgt vor: gegeben einen Graphen $G = (V, E)$ mit $|V| = n$ und eine Kostenfunktion $c : E \to \mathbb{R}_{\geq 0}$ setzen wir zu Beginn $V_1 = \{s\}$, $E_1 = \emptyset$. Für $i = 2, \ldots, n$ sei $v_i \in V \setminus V_i$ der Knoten mit minimalen Anschlusskosten an (V_i, E_i) und $\{u_i, v_i\}$ eine Kante die diese Anschlusskosten realisiert. Dann setzen wir $V_{i+1} = V_i \cup \{v_i\}$ und $E_{i+1} = E_i \cup \{\{u_i, v_i\}\}$. Der Algorithmus antwortet mit (V_n, E_n). Auf diese Weise verwalten wir einen wachsenden Baum der Teilgraph von G ist.

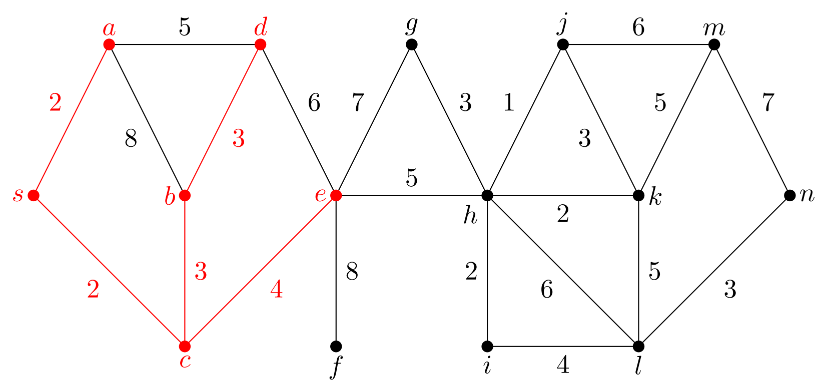


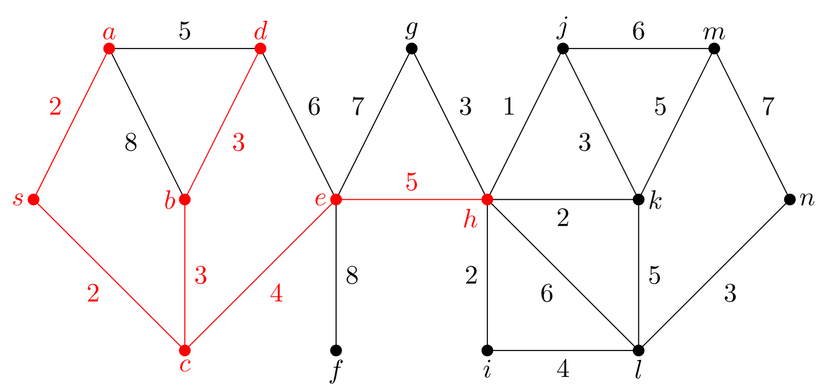


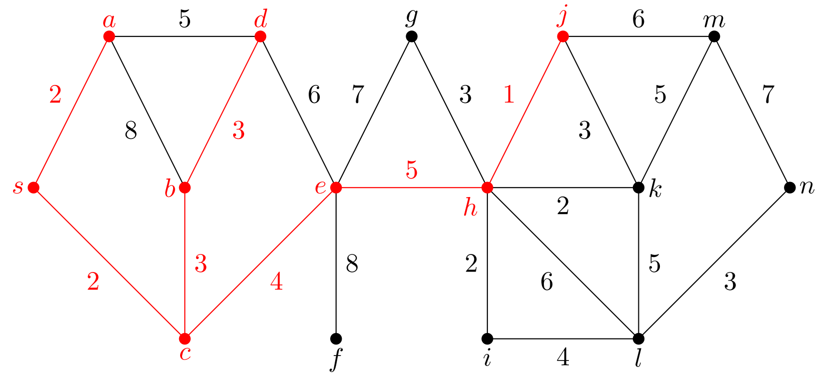


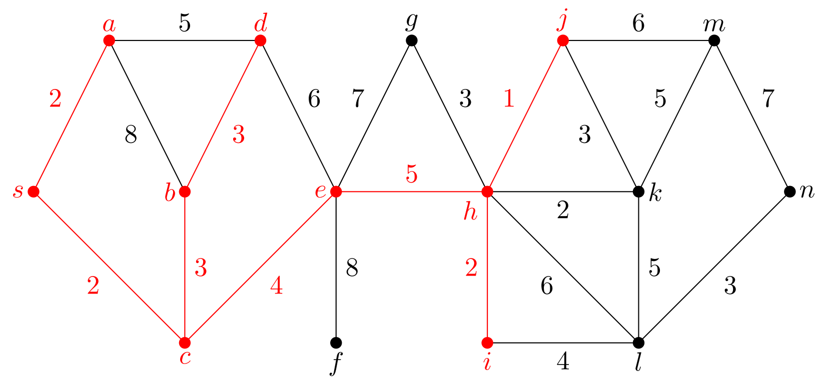

a
5
d
g
j
6
m
2
8
3
6
7
3
1
3
5
7
s
b
e
5
h
2
k
n
3
4
8
2
6
5
3
c
f
i
4
l

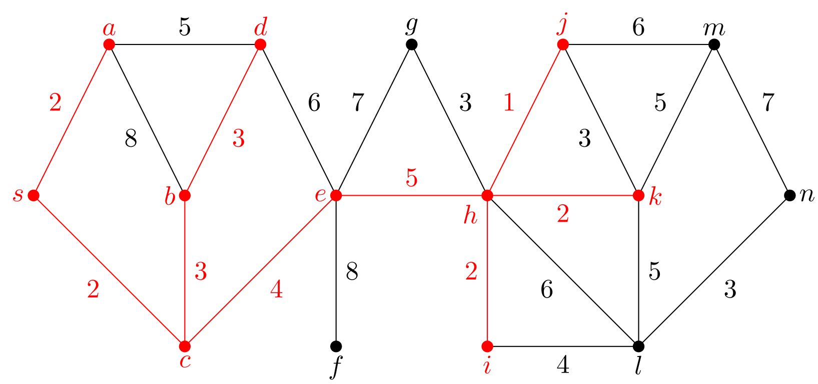

a
5
d
g
j
6
m
2
8
3
6
7
3
1
3
5
7
s
b
e
5
h
2
k
n
3
4
8
2
6
5
3
c
f
i
4
l

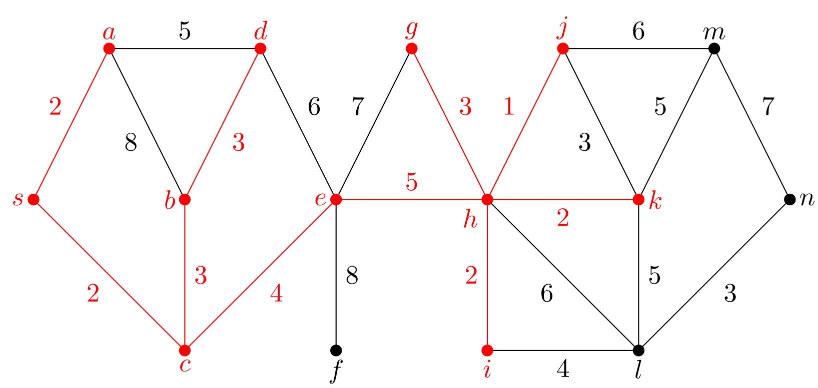

a
5
d
g
j
6
m
2
8
3
6
7
3
1
3
5
7
s
b
e
5
h
2
k
n
3
4
8
2
6
5
3
c
f
i
4
l

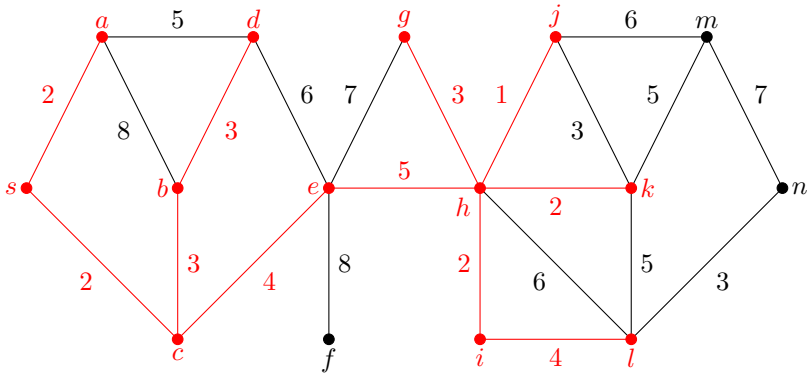

a
5
d
g
j
6
m
2
8
3
6
7
3
1
3
5
7
s
b
e
5
h
2
k
n
3
4
8
2
6
5
3
c
f
i
4
l

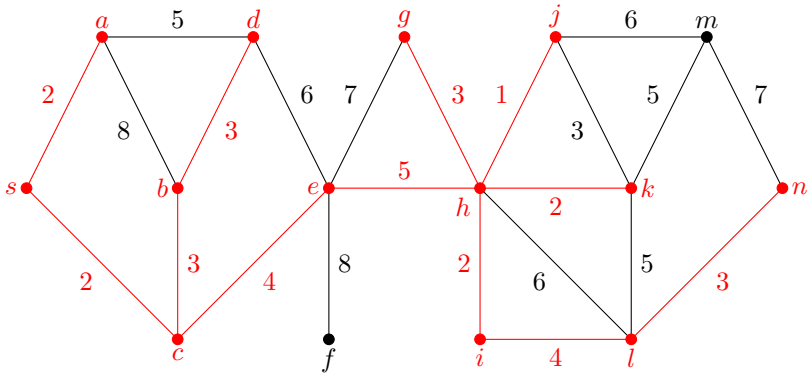

a 5 d g j 6 m
2 8 3 6 7 3 1 3 5 7
s b e 5 h 2 k n
2 3 4 8 2 6 5 3
c f i 4 l

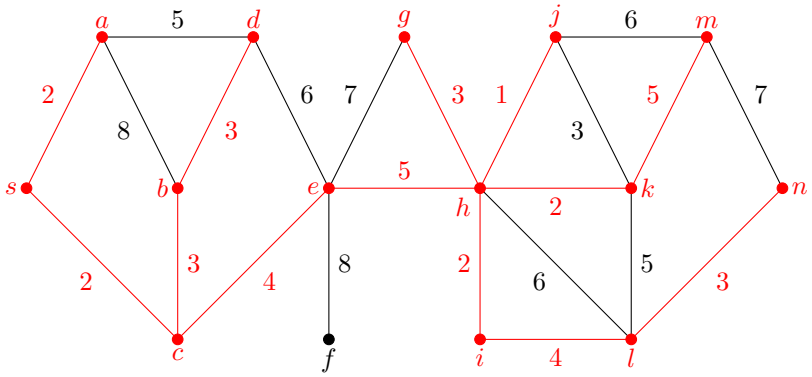

a 5 d g j 6 m
2 8 3 6 7 3 1 3 5 7
s b e 5 h 2 k n
2 3 4 8 2 6 5 3
c f i 4 l

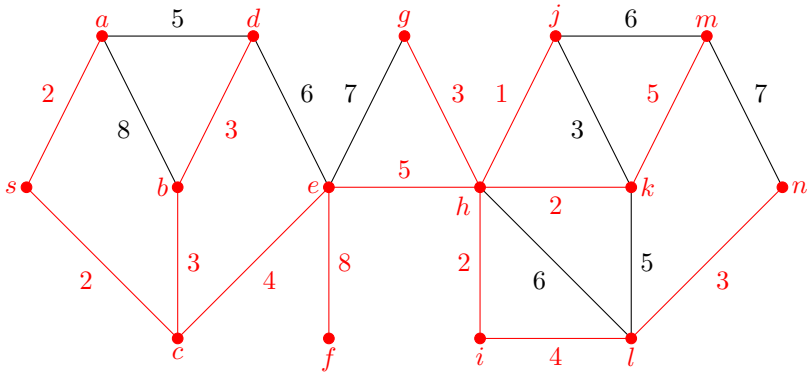

a 5 d g j 6 m
2 8 3 6 7 3 1 3 5 7
s b e 5 h 2 k n
2 3 4 8 2 6 5 3
c f i 4 l


a 5 d g j 6 m
2 8 3 6 7 3 1 3 5 7
s b e 5 h 2 k n
2 3 4 8 2 6 5 3
c f i 4 l


a 5 d g j 6 m
2 8 3 6 7 3 1 3 5 7
s b e 5 h 2 k n
2 3 4 8 2 6 5 3
c f i 4 l


a 5 d g j 6 m
2 8 3 6 7 3 1 3 5 7
s b e 5 h 2 k n
2 3 4 8 2 6 5 3
c f i 4 l


a 5 d g j 6 m
2 8 3 6 7 3 1 5 7 3
s b e 5 h 2 k n
2 3 4 8 2 6 5 3
c f i 4 l


a 5 d g j 6 m
2 8 3 6 7 3 1 5 7 3
s b e 5 h 2 k n
2 3 4 8 2 6 5 3
c f i 4 l

□

Aufgabe 39. Alternative Algorithmen zur Bestimmung minimaler Spannbäume. Nachfolgend sind die Pseudocodes von drei verschiedenen Algorithmen angegeben. Alle drei erhalten als Eingabe einen zusammenhängenden kantenbewerteten Graphen und geben eine Kantenmenge T zurück. Beweisen oder widerlegen Sie für jeden der drei Algorithmen die Behauptung, dass T in jedem Fall ein minimaler Spannbaum ist.

Maybe-MST-A(G, w)
sortiere die Kanten in nichtsteigender Reihenfolge nach ihren Gewichten w
$T := E$
for $e \in E$ (in der soeben berechneten Reihenfolge) **do**
  if $T \setminus \{e\}$ ist ein zusammenhängender Graph **then**
    $T := T \setminus \{e\}$
  end if
end for
return T

Maybe-MST-B(G, w)
$T := \emptyset$
for $e \in E$ (in einer beliebigen Reihenfolge) **do**
  if $T \cup \{e\}$ kreisfrei **then**
    $T := T \cup \{e\}$
  end if
end for
return T

Maybe-MST-C(G, w)
$T := \emptyset$
for $e \in E$ (in einer beliebigen Reihenfolge) **do**
  $T := T \cup \{e\}$
  if T enthält einen Zyklus c **then**
    sei e_0 eine Kante von c mit maximalem Gewicht
    $T := T \setminus \{e_0\}$
  end if
end for
return T

Lösung.

(A) Der Algorithmus von Kruskal startet leer und bezieht nur die nötigsten Kanten (d.h. jene mit minimalen Kosten) in den potentiellen MST mit ein. Unser Algorithmus startet jedoch voll und entfernt die „unnötigsten“ Kanten (d.h. jene mit maximalen Kosten) von dem potentiellen MST.

Die Vermutung, dass Letzterer korrekt ist, klingt daher durchaus plausibel. Um dies zu beweisen, orientieren wir uns entsprechend am Beweis von Satz 7.2.

Sei $e_1, \dots, e_n$ eine Sortierung von E, so dass $c(e_1) \geq \cdots \geq c(e_n)$, sei $E_0 = \emptyset$ und

$$E_{i+1} = \begin{cases} E_i \setminus \{e_{i+1}\} & \text{falls } (V, E_i \setminus \{e_{i+1}\}) \text{ zusammenhängenden ist und} \\ E_i & \text{sonst.} \end{cases}$$

Der Algorithmus antwortet mit (V, E_n).

Beweis. Zunächst ist klar, dass $B = (V, E_n)$ zusammenhängend ist. B ist aber auch zyklenfrei: Angenommen, B wäre nicht zyklenfrei, sei $e_{i_1}, \dots, e_{i_k} \in E_n$ ein Zyklus in B. $e_{i_1}, \dots, e_{i_k}$ wurden vom Algorithmus nicht entfernt, also waren $(V, E_{i_1 - 1} \setminus \{e_{i_1}\}), \dots, (V, E_{i_k - 1} \setminus \{e_{i_k}\})$ nicht zusammenhängend. Sei $\ell = 1, \dots, k$. Weil nun $E_{i_\ell - 1} \setminus \{e_{i_\ell}\} \supset E_n \setminus \{e_{i_\ell}\}$, wäre somit aber $(V, E_n \setminus \{e_{i_\ell}\})$ ebenso nicht

zusammenhängend. Aus dem Zyklus $e_{i_1}, \ldots, e_{i_k} \in E_n$ darf man aber eine beliebige Kante löschen und der resultierende Graph $(V, E_n \setminus \{e_{i_\ell}\})$ wäre noch immer zusammenhängend. Widerspruch!

Für die Minimalität von B reicht es zu zeigen, dass für alle $i \in \{0, \ldots, n\}$ ein minimaler Spannbaum von G mit Kantenmenge M_i existiert so dass $E_i \supseteq M_i$. Dann ist nämlich $E_n = M_n$ und damit B minimal.

Wir gehen mit Induktion nach i vor. Der Fall $i = 0$ ist trivial. Für den Induktionsschritt definieren wir $M_{i+1} = M_i$ falls keine Kante weggenommen wird oder die weggenommene Kante $e_{i+1} \notin M_i$. Sei nun also $E_{i+1} = E_i \setminus \{e_{i+1}\}$, (V, E_{i+1}) zusammenhängend und $e_{i+1} \in M_i$. Weil (V, M_i) als Baum minimal zusammenhängend ist, ist $(V, M_i \setminus \{e_{i+1}\})$ nicht mehr zusammenhängend. ¡¡¡¡¡¡¡ HEAD Weil E_{i+1} zusammenhängend ist, können die beiden ZHKs von $(V, E_i \setminus \{e_{i+1}\}) \supseteq (V, M_i \setminus \{e_{i+1}\}) = (V, E_{i+1})$ durch ein $e_j \in E_{i+1}$ verbunden werden. Dabei ist $e_{i+1} \neq e_j \in E_{i+1} = E_i \setminus \{e_{i+1}\}$. Ebenso ist $e_j \notin M_i \setminus \{e_{i+1}\}$, weil die ZHKs sonst bereits verunden gewesen wären. ======= Weil E_{i+1} zusammenhängend ist, können die beiden Zusammenhangskomponenten von $(V, M_i \setminus \{e_{i+1}\}) \subseteq (V, E_i \setminus \{e_{i+1}\}) = (V, E_{i+1})$ durch ein $e_j \in E_{i+1}$ verbunden werden mit $e_j \neq e_{i+1}$. Ebenso ist $e_j \notin M_i \setminus \{e_{i+1}\}$, weil die Zusammenhangskomponenten sonst bereits verbunden gewesen wären. ¿¿¿¿¿¿¿ b97c5c8db51b80b26a521fa55f79d1dc32d83e29 Daher muss $e_j \notin M_i$. (Oder: Da (V, E_{i+1}) zusammenhängend ist muss e_{i+1} aus einem Zykel z von (V, E_i) stammen. Außerdem ist M_i zyklenfrei und da $e_{i+1} \in M_i$ muss es schon ein e_j aus z geben mit $e_j \notin M_i$.)

$$M_{i+1} := (\underbrace{\underbrace{M_i}_{\subseteq E_{i+1}} \setminus \{e_{i+1}\}}_{\subseteq E_{i+1}}) \dot\cup \underbrace{\{e_j\}}_{\subseteq E_{i+1}} \subseteq E_{i+1}$$

(V, M_{i+1}) ist also zusammenhängend.

Weiters ist $|M_{i+1}| = |M_i| = |V| - 1$ da ja $e_j \notin M_i$ und $e_{i+1} \in M_i$ ist und (V, M_{i+1}) mit Satz 2.3 also ein Spannbaum von G.

Außerdem ist $c(e_j) \leq c(e_{i+1})$, denn $c(e_j) > c(e_{i+1})$ impliziert $j < i+1$ und damit $j-1 < j \leq i < i+1$. Weil die E-Kantenmengen monoton nicht-steigen, heißt das $E_{j-1} \supseteq E_j \supseteq E_i \supseteq E_{i+1}$.

$$\implies E_{j-1} \setminus \{e_j\} \overset{!}{\neq} E_j \supseteq E_{i+1} \ni e_j$$

Laut Konstruktion, und weil $e_j \notin M_i$, wäre dann aber

$$\begin{aligned}(V, E_{j-1} \setminus \{e_j\}) \text{ nicht zusammenhängend } &\supseteq (V, E_i \setminus \{e_j\}) \text{ nicht zusammenhängend} \\ &\supseteq (V, M_i \setminus \{e_j\}) \text{ nicht zusammenhängend } = (V, M_i) \text{ nicht zusammenhängend.}\end{aligned}$$

Widerspruch! Also ist $c(M_{i+1}) \leq c(M_i)$ und, da M_i minimal ist, $c(M_{i+1}) = c(M_i)$ und M_{i+1} also ebenfalls ein minimaler Spannbaum.

Q.E.D.

(B) Maybe-MST-B ist genau der Algorithmus von Kruskal ohne Sortierung.

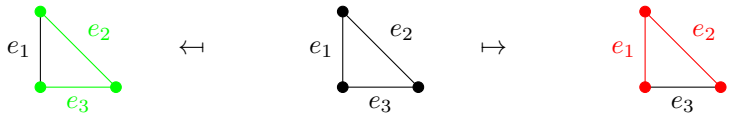
Wir wollen jetzt einen Korrektheitsbeweis des Algorithmus von Kruskal angeben. Der Algorithmus geht wie folgt vor: sei $e_1, \dots, e_n$ eine Sortierung von E so dass $c(e_1) \leq \dots \leq c(e_n)$, sei $E_0 = \emptyset$ und

$$E_{i+1} = \begin{cases} E_i \cup \{e_{i+1}\} & \text{falls } (V, E_i \cup \{e_{i+1}\}) \text{ zyklenfrei ist und} \\ E_i & \text{sonst.} \end{cases}$$

Der Algorithmus antwortet mit (V, E_n).

Ohne Sortierung wird es aber schwer. Dazu betrachten wir ein Dreieck mit einer bösen Kanten-Reihenfolge.

$$E := [e_1, e_2, e_3], \quad c(e_1) > c(e_2) > c(e_3)$$



Der linke (grüne) Spannbaum hat B_l geringere Kosten als der rechte (rote) B_r.

$$c(B_\text{l}) = c(e_2) + c(e_3) < c(e_1) + c(e_2) = c(B_\text{r})$$

(C) Sei $E = [e_1, \dots, e_n]$ eine beliebige Kanten-Reihenfolge. Der gegebene Algorithmus operiert wie folgt.

$$E_0 := \emptyset, \quad E_{i+1} := \begin{cases} (E_i \cup \{e_{i+1}\}) \setminus \{e_{i_1}\}, & \exists\, (e_{i_1}, \dots, e_{i_k}) \text{ Zyklus } \subset (V, E_i \cup \{e_{i+1}\}) : \\ & c(e_{i_1}) \geq c(e_{i_2}), \dots, c(e_{i_k}), \\ E_i \cup \{e_{i+1}\}, & \text{sonst} \end{cases}$$

Der Algorithmus antwortet mit (V, E_n).

Um die Korrektheit des Algorithmus zu beweisen, brauchen wir zunächst ein Lemma.

Lemma. Sei $G = (V, E)$ ein endlicher, zyklenfreier Graph. Sei e eine Kante, sodass $G' = (V, E \cup \{e\})$ einen Zyklus enthält. Dann enthält G' höchstens (d.h. genau) einen Zyklus.

Beweis. Angenommen, G' enthielte 2 verschiedene Zyklen $a = (a_1, \dots, a_n)$ und $b = (b_1, \dots, b_m)$. Offenbar muss $e \in a$, da sonst a Zyklus von G wäre. Dasselbe gilt für $b \ni e$. Nun können wir aber aus $a \setminus e$ und $b \setminus e$ einen Zyklus von G erzeugen. Widerspruch!

Q.E.D.

Bemerkung. Wenn man aus einem Graphen (z.B. G' aus dem obigen Lemma), der genau einen Zyklus besitzt eine Zyklus-Kante löscht, dann ist der resultierende Graph zyklenfrei.

Beweis (Korrektheit von Maybe-MST-C). Laut dem obigen Lemma und der Bemerkung, ist klar, dass $B = (V, E_n)$ zyklenfrei ist. B ist aber auch zusammenhängend:

Sei $i = 1, \dots, n$ und verbinde e_{i+1} zwei ZHKs $Z_1, Z_2 \subseteq (V, E_i)$. Wir behaupten, dass $(V, E_i \cup \{e_{i+1}\})$ zyklenfrei ist. Laut dem obigen Lemma und der Bemerkung, ist nämlich klar, dass (V, E_i) zyklenfrei ist. Daher sind insbesondere die ZHKs (Bäume) Z_1 und Z_2 zyklenfrei. $Z := Z_1 \cup \{e_i\} \cup Z_2 \subseteq (V, E_i \cup \{e_{i+1}\})$ ist also auch zyklenfrei. $(V, E_i \cup \{e_{i+1}\})$ ist also insgesamt Zyklenfrei. Der Algorithmus entscheidet

sich also für $E_{i+1} = E_i \cup \{e_{i+1}\}$. Zusammenfassend: Wenn e_{i+1} zwei ZHKs verbindet wird e_{i+1} vom Algorithmus nicht entfernt.

Wir zeigen mit Induktion, dass es immer, d.h. im i-ten Schritt, genug Kanten $\in \{e_i, \dots, e_n\}$ gibt, sodass alle ZHKs von (V, E_{i-1}) verbunden werden können. Der Induktionsanfang ist klar, weil $G = (V, E)$ zusammenhängend ist und die ZHKs von (V, E_0) genau die Singleton-Graphen $(\{v\}, \emptyset)$, $v \in V$ sind. Für den Induktionsschritt gebe es solche Kanten im i-ten Schritt. Wenn keine der Kanten darin verwendet wurde, gibt es alle davon auch im $(i+1)$-ten Schritt wieder. Wenn eine (e_i) verwendet wurde, dann wurde sie laut den oberen Überlegungen nicht vom Algorithmus aus $E_i \cup \{e_i\}$ entfernt. Weil dann aber gerade zwei ZHKs von (V, E_{i-1}) verbunden wurde, gibt es in (V, E_i) dann aber genau eine ZHK weniger. Die im i-ten Schritt verwendete Kante e_i wird daher im $(i+1)$-ten (bis n-ten) Schritt garnicht mehr gebraucht, um diese Bereiche im Graphen zu verbinden, weil sie ja bereits, vermöge e_i zusammenhängen.

Der Algorithmus arbeitet alle Kanten $\in E$ ab, also auch all jene, die ZHKs verbinden. Der resultierende Graph B ist daher zusammenhängend.

Wir müssen also nur noch zeigen, dass B minimal ist unter allen Spannbäumen. Dazu, reicht es zu zeigen, dass

$$\forall\, i = 1, \dots, n : \forall\, G_Z = (V_Z, E_Z) \text{ ZHK von } (V, \bar{E}_i) : (V_Z, E_Z \cap E_i) \subset \text{ MST von } G_Z.$$

Dabei sei $\bar{E}_i := \{e_1, \dots, e_n\}$ die Menge der Kanten, die der Algorithmus bis zum i-ten Schritt „gesehen“ hat. Weil ja $G = (V, E)$ zusammenhängend ist, hat G nur eine ZHK (sich selbst). Weil $\bar{E}_n = E$, steht für $i = n$ die Behauptung oben praktisch schon da. Wir führen Induktion nach dem i-ten Schritt. Der Induktionsanfang $i = 0$ ist trivial.

Für den Induktionsschritt unterscheiden wir 2 Fälle.

1. Fall (e_{i+1} verbindet 2 ZHK von $(V, \bar{E}_i)$):
   Seien $Z_1 = (V_{Z_1}, E_{Z_1})$ und $Z_2 = (V_{Z_2}, E_{Z_2})$ jene ZHKs.

   $$Z := (V_Z, E_Z) = (V_{Z_1} \cup V_{Z_2}, E_{Z_1} \cup E_{Z_2} \cup \{e_{i+1}\})$$

   Z ist eine ZHK von $(V, \bar{E}_{i+1})$.
   e_{i+1} ist die einzige Verbindung zwischen Z_1 und Z_2. Jeder ST T von Z muss durch Z_1 und Z_2. Also muss T auch durch e_{i+1} gehen, weil sonst wäre T nicht zusammenhängend.

   $$\begin{aligned} \overset{\text{IV}}{\Longrightarrow} \tilde{M}_1 &:= (V_{Z_1}, E_i \cap E_{Z_1}) \subseteq M_1 \text{ MST von } Z_1, \\ \tilde{M}_2 &:= (V_{Z_2}, E_i \cap E_{Z_2}) \subseteq M_2 \text{ MST von } Z_2 \end{aligned}$$

   Sei M MST von Z, dann sind $M_1, M_2 \subseteq M$.

   $$\Longrightarrow \tilde{M}_1, \tilde{M}_2 \subseteq M, \quad e_{i+1} \in M$$

   $$\begin{aligned} \Longrightarrow (V_Z, E_{i+1} \cap E_Z) &= (V_{Z_1} \cup V_{Z_2}, (E_i \cap (E_{Z_1} \cup E_{Z_2})) \cup \{e_{i+1}\}) \\ &= (V_{Z_1} \cup V_{Z_2}, (E_i \cap E_{Z_1}) \cup (E_i \cap E_{Z_2})) \cup \{e_{i+1}\}) \\ &= \tilde{M}_1 \cup \tilde{M}_2 \cup_{\text{Kanten}} \{e_{i+1}\} \\ &\subseteq M \end{aligned}$$

   Für die restlichen ZHKs (außer Z) kann man die IV anwenden.

2. Fall (e_{i+1} verbindet 2 Knoten $\in$ ZHK von $(V, \bar{E}_i)$):
   Sei $Z = (V_Z, E_Z)$ jene ZHK.

   2.1. Fall ($(V, E_i \cup \{e_{i+1}\})$ hat einen Zyklus):
   Sei $z = (e_{i_1}, \ldots, e_{i_k}) \subseteq (V, E_i \cup \{e_{i+1}\}$ jener Zyklus und $c(e_{i_1}) \geq c(e_{i_2}), \ldots, c(e_{i_k})$. Der Zyklus entsteht dort, wo e_{i+1} hinzugefügt wurde. Weil Z als ZHK zusammenhängend ist, ist $e_{i+1} \in z \subseteq (V_Z, E_Z \cap (E_i \cup \{e_{i+1}\}))$. Der Algorithmus entscheidet sich also für $E_{i+1} = (E_i \cup \{e_{i+1}\}) \setminus \{e_{i_1}\}$.

   2.1.1. Fall ($i_1 = i + 1$):

   $$\implies E_{i+1} = (E_i \cup \{e_{i+1}\}) \setminus \{e_{i+1}\} = E_i$$

   $Z' := Z \cup_{\text{Kanten}} \{e_{i+1}\}$ ist eine ZHK von $(V, \bar{E}_{i+1})$ Der MST von $Z \subseteq (V, \bar{E}_i)$ ist derselbe wie der von Z'. Die IV ist also direkt anwendbar.

   2.1.2. Fall ($i_1 \neq i + 1$):

   $$\implies c(i_1) \geq c(i + 1)$$

   Der MST M_i von $(V_Z, E_Z \cap \bar{E}_i)$ aus der IV verläuft entlang den Kanten $z \setminus \{e_{i+1}\} \subseteq (V, E_i)$. Der MST M_{i+1} von $(V_{Z'}, E_{Z'} \cap \bar{E}_{i+1})$ verläuft genau gleich, bloß durch die billigere Kante e_{i+1} statt der teureren e_{i_1}.

   2.2. Fall ($(V, E_i \cup \{e_{i+1}\})$ ist zyklenfrei):
   Der Algorithmus entscheidet ist daher für $E_{i+1} = E_i \cup \{e_{i+1}\}$.

   $$\implies (V, E_{i+1}) \text{ zyklenfrei} \implies (V_Z, E_Z \cap E_{i+1}) \text{ zyklenfrei}$$

   Weil Z als ZHK zusammenhängend ist, gehört e_{i+1} zu einem Zyklus $z_1 = (e_{i_1}, \ldots, e_{i_k})$. Sei $e_{i_\ell} \in E_Z \cap (\bar{E}_i \setminus E_i)$ eine, von dem Algorithmus weggeschnittene, Kante. e_{i_ℓ} wurde weggeschnitten, weil sie maximale Kosten in einem, von z verschiedenen Zyklus $z' \subseteq (V, E_{i_\ell - 1} \cup \{e_{i_\ell}\})$ hatte. z' grenzt an 1 (sogar 2) Knoten $\in V_Z$ an. Weil Z als ZHK zusammenhängend ist, ist $z' \subseteq Z$ ganz in Z enthalten. z_2 bildet nun wieder einen Zyklus.

   $$z_2 := (z_1 \setminus \{e_{i_\ell}\}) \cup \underbrace{(z' \setminus \{e_{i_\ell}\})}_{\subseteq (V_Z, E_Z \cap E_{i_\ell - 1})} = (z_1 \cup z') \setminus \{e_{i_\ell}\}$$

   Es kann nun sein, dass der Austausch von e_{i_ℓ} mit dem Umweg $z' \setminus \{e_{i_\ell}\}$ weitere Kanten liefert, die wiederum ausgetauscht werden müssen (und wieder, ..., und wieder). Die Umwege für den Austausch liegen aber in Graphen, mit immer weniger Kanten. Insofern sind die $z_1, z_2, \ldots$ strikt halbgeordnet (durch $(H, \subsetneq)$). (Fun Fact: Das Hasse-Diagrmm dieser HO ist sogar ein Baum mit Wurzel z_1.)
   Gegebenenfalls wenden wir auf z_2 also dasselbe Argument wie auf z_1 an (und wieder, ..., und wieder). Weil G endlich ist, gibt es nur endlich viele verschiedene Zyklen. Daher ist insbesondere die Halbordnung H endlich.
   Irgendwann bekommen wir also einen Zyklus z_∞, der nur nicht-weggeschnittene Kanten, d.h. $\in E_Z \cap E_{i+1}$ hat, d.h. minimales Element von H ist. Das ist ein Widerspruch dazu, dass $(V_Z, E_Z \cap E_{i+1})$ zyklenfrei ist!

Q.E.D.

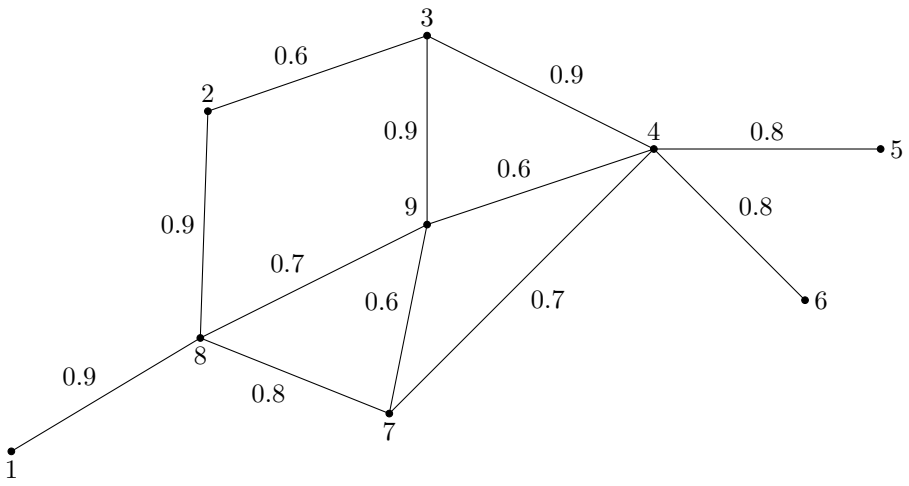
□

Aufgabe 40. Gegeben sei ein gerichteter oder ungerichteter Graph $G = (V, E)$. Jeder Kante $(u, v) \in E$ ist ein reeller Wert $r(u, v)$ mit $0 \leq r(u, v) \leq 1$ zugeordnet, der die Zuverlässigkeit einer Datenübertragung vom Knoten u zum Knoten v darstellt. Wir interpretieren $r(u, v)$ als die Wahrscheinlichkeit, dass der Kanal von u nach v nicht versagt. Weiters setzen wir voraus, dass diese Wahrscheinlichkeiten paarweise unabhängig sind.

a Geben Sie einen effizienten Algorithmus an, der den zuverlässigsten Kanal zwischen zwei gegebenen Knoten bestimmt.

b Wenden Sie diesen Algorithmus auf folgenden Graphen an, um einem möglichst zuverlässigen Kommunikationskanal zwischen dem Knoten 1 und 5 herzustellen, wobei die Wahrscheinlichkeiten $r(u, v)$ bei den entsprechenden Kanten angegeben sind.

Hinweis: Dijkstra Algorithmus.



Lösung. a

Algorithmus 29 Algorithmus von Dijkstra

Prozedur DIJKSTRA(V, E, c, s, t)
  Sei B ein neues Datenfeld der Länge $|V|$, überall mit **falsch** initialisiert
  Für alle $v \in V$
    $v.x := \infty$
    $v.\mathit{Vorgänger} :=$ NIL
  Ende Für
  $s.x := 0$
  $Q :=$ MIN-PRIORITÄTSWARTESCHLANGE(V)
  Solange Q nicht leer
    $v :=$ EXTRAHIEREMINIMUM(Q)
    $B[v] :=$ **wahr**
    Für alle $(v, w) \in E$
      Falls $v.x + c(v, w) < w.x$ **und** $B[w] =$ **falsch dann**
        $w.\mathit{Vorgänger} := v$
        REDUZIEREPRIORITÄT($Q, w, v.x + c(v, w)$)
      Ende Falls
    Ende Für
  Ende Solange
  Antworte $(t, t.\mathit{Vorgänger}, t.\mathit{Vorgänger}.\mathit{Vorgänger}, \ldots, s)^{-1}$
Ende Prozedur

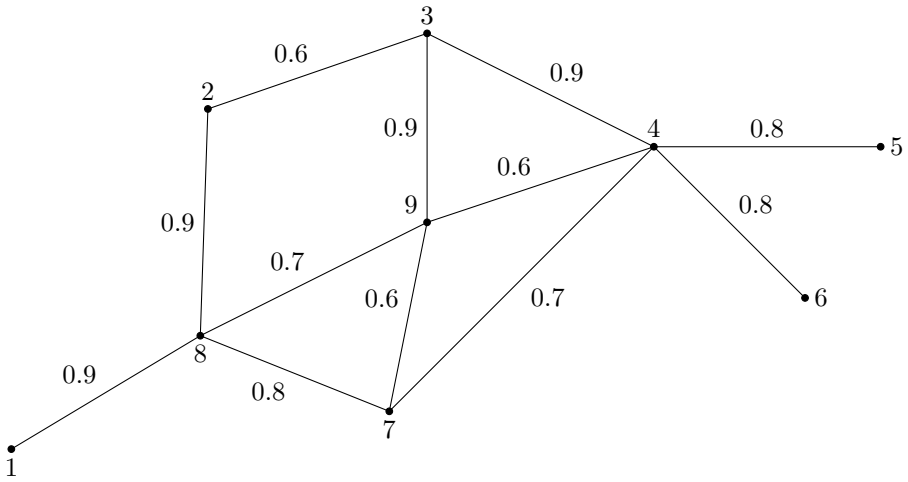
Der Algorithmus von Djkstra „addiert“ die Kanten-Kosten und sucht einen Pfad mit „minimalen“ Pfad-Kosten. Wir wollen aber „multiplizieren“ und den Pfad mit „maximaler“ Sicherheit finden. Daher muss der Algorithmus von Djkstra mit der Kostenfunktion $c := \log \frac{1}{r} = -\log r$ angewendet werden.

$$\begin{array}{ccc} ((0,1),\cdot,\geq) & \overset{1/\cdot}{\cong} & ((1,\infty),\cdot,\leq) \\ \overset{\log}{\cong} & & \overset{\log}{\cong} \\ ((-\infty,0),+,\geq) & \overset{-}{\cong} & ((0,\infty),+,\leq) \end{array}$$

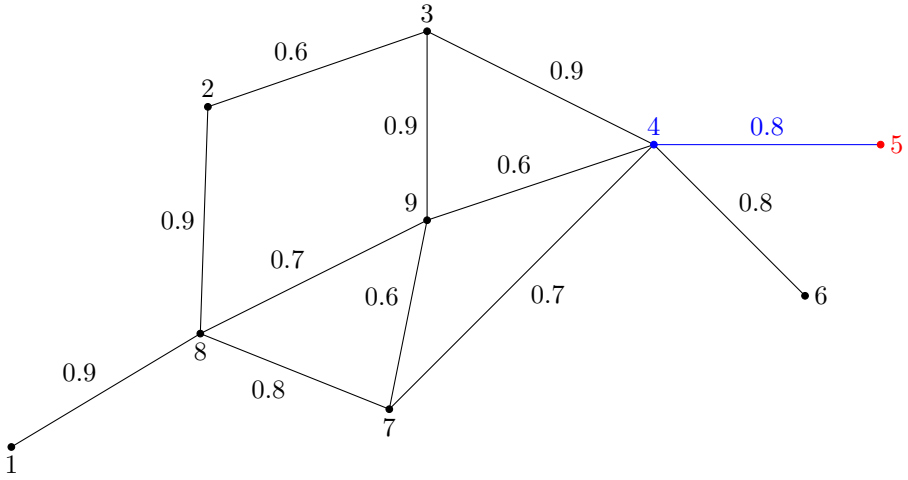
Algorithm 3 Finde verlässlichsten Pfad in $G = (V, E)$ von s nach t

```
1: procedure VerlaesslichsterPfad(V, E, r, s, t)
2:     Sei B ein neues Datenfeld der Länge |V| überall mit falsch initialisiert
3:     for alle v ∈ V do
4:         v.x := 0
5:         v.Vorgaenger := NIL
6:     end for
7:     s.x := 1
8:     Q := Max-Prioritätswarteschlange(V)
9:     while Q nicht leer do
10:        v := ExtrahiereMaximum(Q)
11:        B[v] := wahr
12:        for alle (v, w) ∈ E do
13:            if v.x * r(v, w) > w.x and B[w] = falsch then
14:                w.Vorgaenger := v
15:                ErhöhePriorität(Q, w, v.x * r(v, w))
16:            end if
17:        end for
18:    end while
19:    return (t, t.Vorgaenger, t.Vorgaenger.Vorgaenger, ..., s)^-1
20: end procedure
```

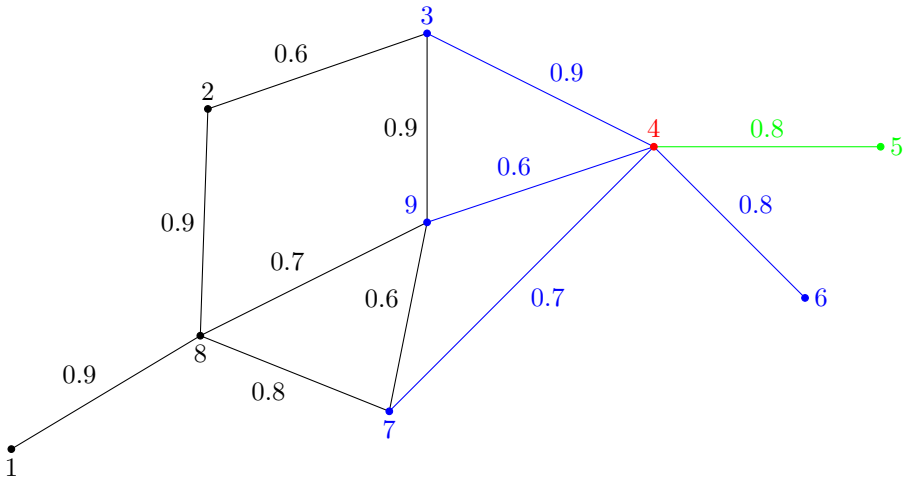
b Wir diskutieren die Schritte und Ergebnisse des Algorithmus von Dijkstra im $((0,1),\cdot,\geq)$-Setting. Genau genommen operiert dieser in der isomorphen geordneten Halbgruppe $((0,\infty),+,\leq)$. Zwecks mangelnder arithmetischer Exaktheit der isomorphen Transformation (vermöge c) der Angabe-Daten, bleiben wir dabei, dies lediglich anzumerken.



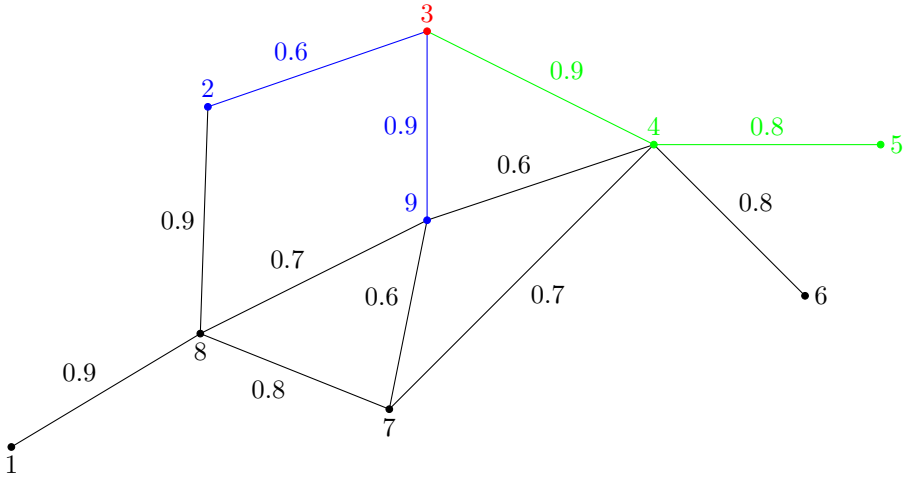
Knoten	1	2	3	4	5	6	7	8	9
Priorität	0	0	0	0	1	0	0	0	0
Vorgänger	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL
Kandidat					$\top$				



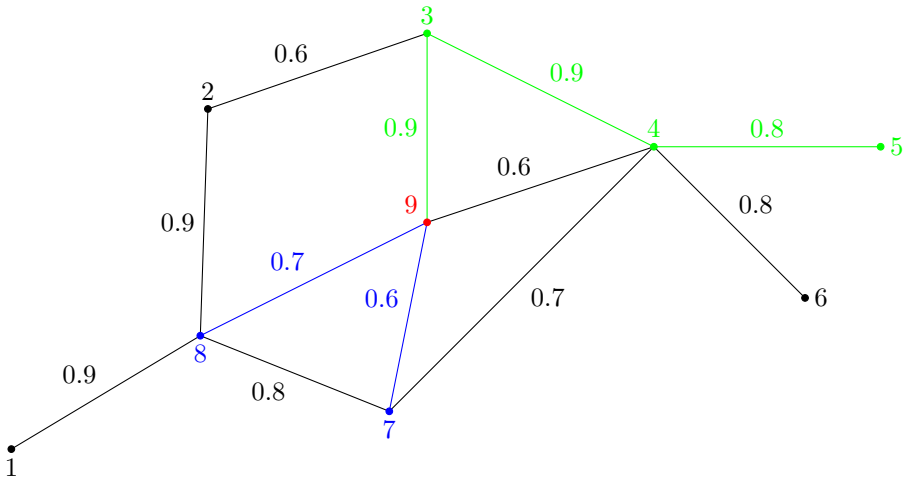
Knoten	1	2	3	4	5	6	7	8	9
Priorität	0	0	0	$0 < 1 \cdot 0.8 = 0.8$	1	0	0	0	0
Vorgänger	NIL	NIL	NIL	5	NIL	NIL	NIL	NIL	NIL
Kandidat				$\top$					



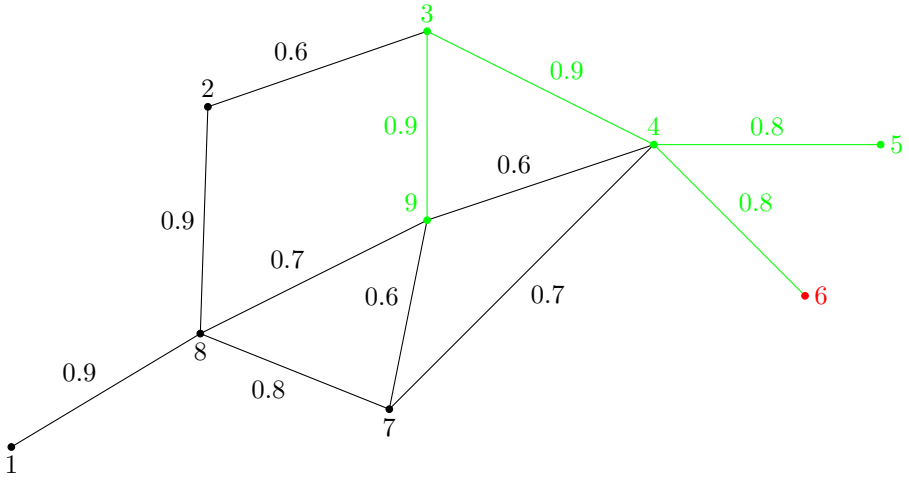
Knoten	1	2	3	4	5	6	7	8	9
Priorität	0	0	$0 < 0.72$	0.8	1	$0 < 0.64$	$0 < 0.56$	0	$0 < 0.48$
Vorgänger	NIL	NIL	4	5	NIL	4	4	NIL	4
Kandidat			$\top$			$\bot$	$\bot$		$\bot$



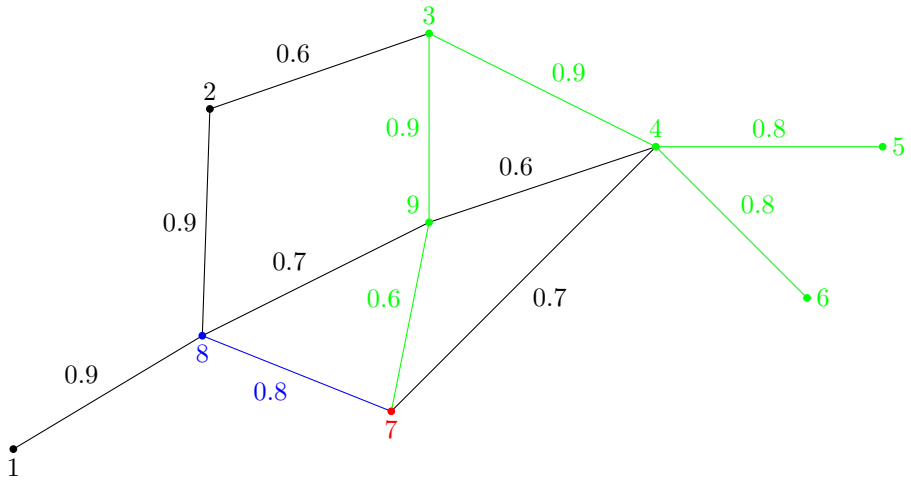
Knoten	1	2	3	4	5	6	7	8	9
Priorität	0	$0 < 0.72 \cdot 0.6 = 0.432$	0.72	0.8	1	0.64	0.56	0	$0.48 < 0.72 \cdot 0.9 = 0.648$
Vorgänger	NIL	3	4	5	NIL	4	4	NIL	3
Kandidat		$\bot$				$\bot$	$\bot$		$\top$



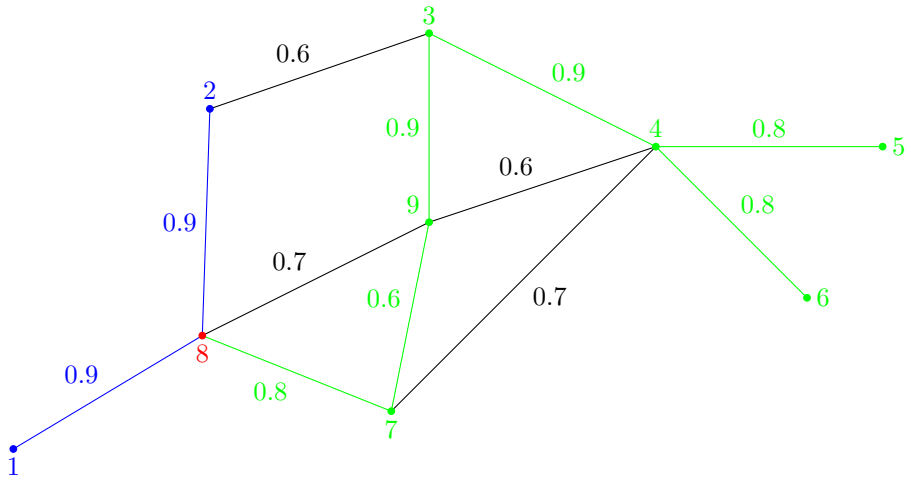
Knoten	1	2	3	4	5	6	7	8	9
Priorität	0	0.432	0.72	0.8	1	0.64	$0.56 \not< 0.648 \cdot 0.6 = 0.3888$	$0 < 0.648 \cdot 0.7 = 0.4536$	0.648
Vorgänger	NIL	3	4	5	NIL	4	4	9	3
Kandidat		$\bot$				$\top$	$\bot$	$\bot$	



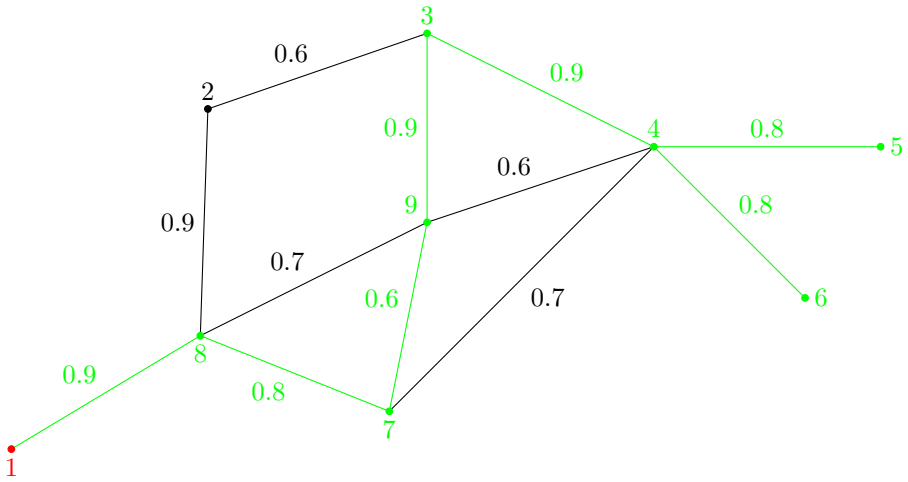
Knoten	1	2	3	4	5	6	7	8	9
Priorität	0	0.432	0.72	0.8	1	0.64	0.56	0.4536	0.648
Vorgänger	NIL	3	4	5	NIL	4	4	9	3
Kandidat		$\bot$					$\top$	$\bot$	



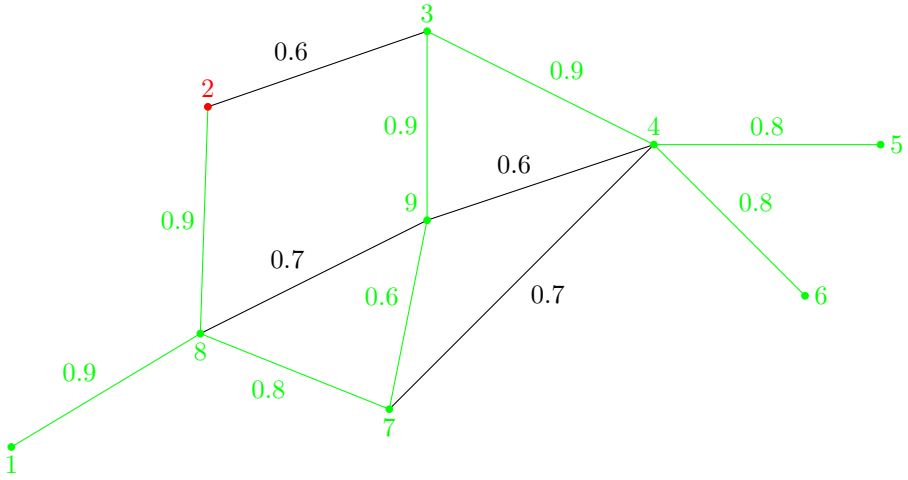
Knoten	1	2	3	4	5	6	7	8	9
Priorität	0	0.432	0.72	0.8	1	0.64	0.56	$0.4536 \not< 0.56 \cdot 0.8 = 0.448$	0.648
Vorgänger	NIL	3	4	5	NIL	4	4	9	3
Kandidat		$\bot$						$\top$	



Knoten	1	2	3	4	5	6	7	8	9
Priorität	$0 < 0.4536 \cdot 0.9 = 0.40824$	$0.432 \not< 0.4536 \cdot 0.9 = 0.40824$	0.72	0.8	1	0.64	0.56	0.4536	0.648
Vorgänger	8	3	4	5	NIL	4	4	9	3
Kandidat	$\top$	$\bot$							



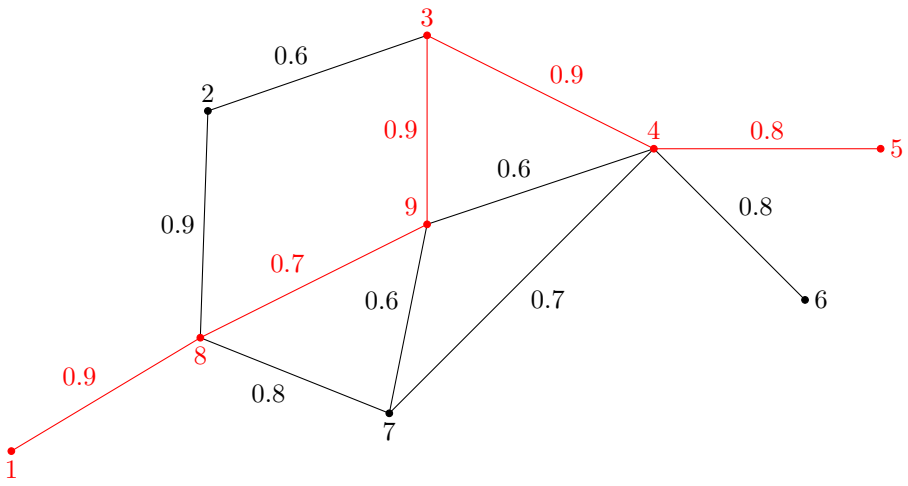
Knoten	1	2	3	4	5	6	7	8	9
Priorität	0.40824	0.432	0.72	0.8	1	0.64	0.56	0.4536	0.648
Vorgänger	8	3	4	5	NIL	4	4	9	3
Kandidat		$\top$							



Knoten	1	2	3	4	5	6	7	8	9
Priorität	0.40824	0.432	0.72	0.8	1	0.64	0.56	0.4536	0.648
Vorgänger	8	3	4	5	NIL	4	4	9	3
Kandidat									

Der resultierende (zuverlässigste) Kommunikationskanal (d.h. Pfad) zwischen 1 und 5 lautet daher

$$(1, 8, 9, 3, 4, 5).$$



□

Aufgabe 41. Beweisen Sie, dass ein Unabhängigkeitssystem (E, S) genau dann ein Matriod ist, wenn für alle $A \subseteq E$ gilt, dass alle maximalen unabhängigen Teilmengen von A gleichmächtig sind.

Lösung.

> **Definition 7.2.** Sei E eine endliche Menge und $\mathcal{U} \subseteq \mathfrak{P}(E)$. $M = (E,\mathcal{U})$ heißt *Matroid* falls:
>
> 1. $\emptyset \in \mathcal{U}$,
> 2. $A \subseteq B \in \mathcal{U}$ impliziert $A \in \mathcal{U}$ und
> 3. es gilt die folgende *Austauscheigenschaft*:
>   Ist $A, B \in \mathcal{U}$ mit $|A| > |B|$, dann gibt es $x \in A \setminus B$ so dass $B \cup \{x\} \in \mathcal{U}$.

Definition (Unabhängigkeitssystem). Ein *Unabhängigkeitssystem* (E, S) ist ein Matriod, ohne geforderter Austauscheigenschaft. Die Elemente von S heißen *unabhängig* und die von $\mathcal{P}(E) \setminus S$ heißen *abhängig*.

1. Richtung („⇒“):

> **Definition 7.3.** Sei $(E,\mathcal{U})$ ein Matroid. Eine Menge $A \in \mathcal{U}$ heißt *Basis* falls kein A' existiert mit $A \subset A' \in \mathcal{U}$.

> **Satz 7.5.** *Sei $M = (E,\mathcal{U})$ ein Matroid, seien B_1, B_2 Basen von M, dann ist $|B_1| = |B_2|$.*

Sei (E, S) ein Matriod und $A \subseteq E$.

$$S_A := \{B \in S : B \subseteq A\}$$

Offensichtlich ist (A, S_A) ein Matriod. Die Behauptung folgt daher aus Satz 7.5. Dieser hat einen sehr kurzen Beweis, der wie folgt aussieht.

Nehmen wir an es wären $B, C \in S$ zwei maximal linear unabhängige Teilmengen von A mit $|B| \neq |C|$. Ohne Beschränkung der Allgemeinheit nehmen wir $|B| > |C|$ an. Dann gibt es ein $x \in (B \setminus C) \subseteq A$ mit $C \cup \{x\} \in S$, ein Widerspruch zur Maximalität von C.

2. Richtung („$\Leftarrow$“):

   Seien $A, B \in S$ und $|A| > |B|$. Sei weiters $C := A \cup B$. Es gibt eine S-unabhängige C-maximale Obermenge von A.

$$\implies \exists\, A' \in S : A \subseteq A' \subseteq C \text{ maximal}$$

   $B \subseteq C = A \cup B$ ist nicht maximal, weil sonst wäre wegen der gegebenen Zusatzeigenschaft

$$|B| = |A'| \geq |A| > |B|.$$

   Es gibt daher eine S-unabhängige C-maximale echte Obermenge von B.

$$\implies \exists\, B' \in S : B \subsetneq B' \subseteq C \text{ maximal}$$

   Wenn B zu $B' \subseteq C = A \cup B$ erweitert wird, müssen Elemente aus A hinzugefügt werden, d.h.

$$\begin{aligned} &\implies \emptyset \neq B' \setminus B \subseteq A. \\ &\implies \emptyset \neq B' \setminus B \subseteq A \setminus B \end{aligned}$$

   Sei $x \in B' \setminus B \subseteq B'$, dann ist $B \cup \{x\} \subseteq B' \in S$. Weil S als Unabhängigkeitssystem nach unten $\subseteq$-abgeschlossen ist, folgt $B \cup \{x\} \in S$.

□

Aufgabe 42. Sei $G = (V, E)$ ein ungerichteter Graph und k eine beliebige nichtnegative ganze Zahl. Ferner sei $M_k(G) = (E, S)$. Die Menge S sei definiert als die Menge aller Teilmengen von E, die sich in der Form $M \cup F$ darstellen lassen, wobei M eine höchstens k-elementige Menge bezeichnet und F eine Menge, für die (V, F) ein Wald ist. Beweisen Sie, dass $M_k(G)$ ein Matroid ist.

Lösung.

$$M_k(G) = (E, S), \quad S = \{M \cup F \subseteq E : |M| \leq k, (V, F) \text{ Wald}\}$$

1. Eigenschaft ($\emptyset \in S$):

   Dafür müssen $M, F = \emptyset$. Für M ist das sowieso möglich. Für F geht das auch, weil jeder Graph der Form $(V, \emptyset)$ ja keine Kanten, also keine Zyklen haben kann. Insofern ist $(V, \emptyset)$ zyklenfrei, also ein Wald.

2. Eigenschaft ($\subseteq$-Abgeschlossenheit):

   Sei $A \subseteq B \in S$, dann lässt sich $B = M_B \cup F_B$ schreiben, wobei $|M_B| \leq k$ und (V, F_B) ein Wald (d.h. zyklenfrei) ist. Wenn man $M_A := M_B \cap A$ und $F_A := F_B \cap A$ setzt, erhält man $A \in S$.

$$\begin{aligned} \implies & |M_A| = |M_B \cap A| \leq |M_B| \leq k, \\ & F_A = F_B \cap A \subseteq F_B \implies (V, F_B) \text{ zyklenfrei } \supseteq (V, F_A) \text{ zyklenfrei}, \\ & M_A \cup F_A = (M_B \cap A) \cup (F_B \cap A) = \underbrace{(M_B \cup F_B)}_{B} \cap A = A \\ \implies & A \in S \end{aligned}$$

3. Eigenschaft (Austauscheigenschaft):
   Seien $A, B \in S$ mit $|A| > |B|$, dann lassen sich A und B darstellen als

$$A = M_A \dot\cup F_A, \quad B = M_B \dot\cup F_B$$
$$|M_A|, |M_B| \leq k, \quad (V, F_A), (V, F_B) \text{ zyklenfrei.}$$

   Weil M_A und M_B auch kleiner sein dürfen, haben wir $M \cap F = \emptyset$ gewählt.

   1. Fall ($|M_B| < k$):
      In diesem Fall wählen wir $x \in A \setminus B \neq \emptyset$ beliebig.

$$\implies |M_B \cup \{x\}| \leq |M_B| + 1 \leq k$$
$$\implies B \cup \{x\} = M_B \cup F_B \cup \{x\} = (M_B \cup \{x\}) \cup F_B \in S$$

   2. Fall ($|M_B| = k$):
      Nach Korollar 2.1 besteht der Wald (V, F_A) aus $|V| - |F_A|$ und der Wald (V, F_B) aus $|V| - |F_B|$ Bäumen.

$$\implies |F_A| + |M_A| = |A| > |B| = |M_B| + |F_B| = k + |F_B| \geq |M_A| + |F_B|$$
$$\implies |F_A| > |F_B|$$
$$\implies |V| - |F_A| < |V| - |F_B|$$

      Nach dem Schubfachprinzip existiert 1 Baum T im Wald (V, F_A), dessen Knoten zu mindestens 2 verschiedenen Bäumen T_1 und T_2 im Wald (V, F_B) gehören.
      Wie wird das Schubfachprinzip hier genau angewendet? Die „Gegenstände“, welche wir auf die Schubfächer aufteilen wollen sind die Bäume im Wald (V, F_B). Ein solcher Baum R ist nicht leer, er enthält also mindestens einen Knoten v. Da der Wald (V, F_A) alle Knoten aus V enthält gibt es schon einen Baum R' aus diesem Wald, der v als Knoten enthält. Beschriften wir nun jedes Schubfach mit einem Baum aus dem Wald (V, F_A) so können wir jeden Baum R aus dem Wald (V, F_B) in ein Schubfach legen, das mit einem Baum R' aus dem Wald (V, F_A) beschriftet ist, wobei sich R und R' mindestens einen Knoten Teilen. Da der Wald (V, F_A) aber weniger Bäume hat als der Wald (V, F_B) müssen in mindestens einem Schubfach dann schon mindestens zwei Bäume aus dem Wald (V, F_B) liegen.
      T ist als Baum zusammenhängend. Daher gibt es eine Kante $e = \{v, w\} \in_{\text{Kanten}} T$, sodass $v \in T_1$ und $w \in T_2$ in verschiedenen Bäumen vom Wald (V, F_B) liegen.

$$\begin{aligned}\implies & e \in_{\text{Kanten}} T \subseteq (V, F_A) \subseteq_{\text{Kanten}} = M_A \cup F_A = A, \\ & e \notin_{\text{Kanten}} T_1, T_2 \subseteq (V, F_B) \subseteq_{\text{Kanten}} = M_B \cup F_B = B \\ \implies & e \in A \setminus B\end{aligned}$$

      $T' = T_1 \cup T_2 \cup_{\text{Kanten}} \{e\}$ ist weiterhin zusammenhängend, weil T_1 und T_2 es sind und vermöge e verbunden. T' ist auch zyklenfrei, weil T_1 und T_2 es sind und es keinen Zyklus gibt, der durch e geht. T' ist also ein Baum aus $(V, F_B \cup \{e\})$.
      Die restlichen Bäume daraus sind jene $\neq T_1, T_2$ vom Wald (V, F_B). Daher ist $(V, F_B \cup \{e\})$ zyklenfrei, d.h. ein Wald. Die Kante $e \in A \setminus B$ kann als Austausch-Element verwendet werden.

$$\implies B \cup \{e\} = M_B \cup (F_B \cup \{e\}) \in S$$

□

Diskrete und Geometrische Algorithmen

8. Übung

Richard Weiss Florian Schager Christian Sallinger Fabian Zehetgruber
Paul Winkler Christian Göth

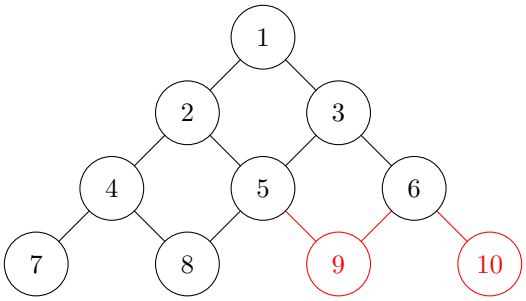
11.01.2021

Aufgabe 43. Ein bi-parental Heap (oder Beap) ist eine Datenstruktur, wo ein Knoten zwei Eltern hat (außer, es handelt sich um den ersten oder letzten Knoten auf einem Level) und zwei Kinder hat (außer, es ist ein Knoten am untersten Level).

a) Sei n die Anzahl der Knoten in einem Beap. Wie groß ist die Höhe des Beaps in etwa? Wie viele Elemente befinden sich auf dem letzten Level (unter der Annahme, dass dieser voll ist)?

b) Betrachten wir nun einen MAX-Beap, d.h. einen Beap, bei dem die Einträge jedes Knotens größergleich den Einträgen in den Kindern dieses Knotens sind. Formulieren Sie einen Pseudocode für die Prozedur MAX-BEAPIFY, wobei das Feld A und der Index i gegeben sind, die Teil-Beaps mit den Wurzeln LEFT$[i]$ und RIGHT$[i]$ MAX-Beaps sind, der Eintrage $A[i]$ aber möglicherweise kleiner als die Einträge in den Kindern sein kann.

Lösung. Folgendes Bild ist in (inklusive rot vollständiger) Beap mit Indizierung, die die Datenfeld-Position des Elements angibt.



a Bezeichne also n die Anzahl der Knoten, h die Höhe und k_i die Anzahl der Knoten auf dem i-ten Level, $i = 1, \dots, h$. Nehmen wir einmal an, dass das letzte Level voll ist. (Das ist bei der gegebenen Definition immer der Fall.) Wir sehen, dass, dass $k_i = i$.

$$n(h) = \sum_{i=1}^{h} i = \frac{h(h+1)}{2} \iff h^2 + h - 2n = 0 \iff h(n) \overset{h\geq 0}{:=} -\frac{1}{2} + \sqrt{\frac{1}{4} + 2n} = \frac{\sqrt{1+8n} - 1}{2}$$

Sei unser Beap mit n Knoten nicht vollständig. Wir können den kleinsten vollständigen Ober-Beap mit $n^+ > n$ Knoten betrachten. Dieser hat klarerweise die selbe Höhe. Wir können aber auch den kleinsten vollständigen Unter-Beap mit $n^- < n$ Knoten betrachten. Dieser hat klarerweise eine Höhe weniger.

$$h(n^-) + 1 = \frac{\sqrt{1+8n^-}-1}{2} + 1 \leq \frac{\sqrt{1+8n}-1}{2}$$
$$\leq h(n) := \left\lceil \frac{\sqrt{1+8n}-1}{2} \right\rceil = h(n^+) = \frac{\sqrt{1+8n^+}-1}{2}$$

In einem vollständigen Beap, mit n Knoten, ist die Anzahl der Knoten im letzten Level genau $h(n)$.

b Wir orientieren uns an der Prozedur ErzeugeHalde aus dem Skript. Wir nehmen also an, dass der Beap als Datenfeld realisiert ist und wir Identifizieren die Elemente mit ihrer Position in dem Datenfeld. Dabei gehen wir wie bei den Halden vor. Zunächst definieren wir uns eine Funktion sowie einige Prozeduren.

$$\textsc{Level}(i) := \left\lceil \frac{\sqrt{1+8i}-1}{2} \right\rceil, \quad \tilde{h}(i) := \frac{\sqrt{1+8i}-1}{2}$$

Wir bezeichnen (sofern vorhanden) den rechten Elternteil als Vater, und den linken Elternteil als Mutter, sowie das rechte Kind als Sohn, und das linke Kind als Tochter.

```
procedure Mutter(i)
    if Level(i) − 1 = h̃(i − 1) then    ▷ Es gibt keine Mutter; das Element ist ganz links im Level.
        return NIL
    else
        return i − Level(i)
    end if
end procedure
```

```
procedure Vater(i)
    if Level(i) = h̃(i) then            ▷ Es gibt keinen Vater; das Element ist ganz rechts im Level.
        return NIL
    else
        return i − Level(i) + 1
    end if
end procedure
```

```
procedure Tochter(A, i)
    if i + Level(i) > A.HLänge then                    ▷ Es gibt keine Tochter.
        return NIL
    else
        return i + Level(i)
    end if
end procedure
```

```
procedure Sohn(A, i)
    if i + Level(i) + 1 > A.HLänge then                ▷ Es gibt keinen Sohn.
        return NIL
    else
```

```
        return i + Level(i) + 1
    end if
end procedure
```

Algorithmus 22 Extraktion des Maximums aus Prioritätswarteschlange

```
Prozedur ExtrahiereMaximum(Q)
    Falls Q.HLänge = 0 dann
        Antworte "Q ist leer"
    sonst
        max := Q[1]
        Q[1] := Q[Q.HLänge]
        Q.HLänge := Q.HLänge − 1
        AbwärtsKorrigieren(Q, 1)
        Antworte max
    Ende Falls
Ende Prozedur

Prozedur AbwärtsKorrigieren(Q, i)
    Solange i ≤ Q.HLänge
        Sei m ∈ {i, links(i), rechts(i)} ∩ [1, ..., Q.HLänge] so dass Q[m].x maximal ist
        Falls m = i dann                                   ▷ Abwärts-Korrektur beendet
            i := Q.HLänge + 1
        sonst
            Vertausche Q[i] und Q[m]
            i := m
        Ende Falls
    Ende Solange
Ende Prozedur
```

Nun können wir unsere Prozendur mittels AbwärtsKorrigieren realisieren, wobei man die Prozedueren Links und Rechts durch Tochter bzw. Sohn ersetzen muss.

```
procedure Max-Beapify(A, i)
    A.HLänge := A.Länge
    Vertausche A[1] und A[i]
    k := Vater(A.HLänge)
    if k = NIL then
        k := Mutter(A.HLänge)
    end if
    if Level(A.HLänge) > 2 then
        for j = k, ..., 2 do
            AbwärtsKorrigieren(A, j)
        end for
    end if
end procedure
```

□

Aufgabe 44. Wir können einen Heap bauen, indem wir wiederholt die in der Vorlesung kennengelernte Prozedur EUNFÜGEN aufrufen, um ein Element in den Heap einzufügen. Betrachten Sie die wie folgt geänderte ERZEUGE-HALDE-Prozedur:

```
ERZEUGE-HALDE'(A)
```

A.heap-size := 1
for $i = 2$ to A.länge **do**
  EUNFÜGEN($A, A[i]$)
end for

a Erzeugen die Prozeduren ERZEUGE-HALDE und ERZEUGE-HALDE' immer denselben Heap, wenn sie auf das gleiche Eingabefeld angewendet werden? (Beweis oder Gegenbeispiel)

b Zeigen Sie, dass ERZEUGE-HALDE' im worst case eine Laufzeit von $\Theta(n \log n)$ benötigt, um einen Heap mit n Elementen zu erzeugen.

Lösung.

Algorithmus 20 Einfügen in eine Prioritätswarteschlange

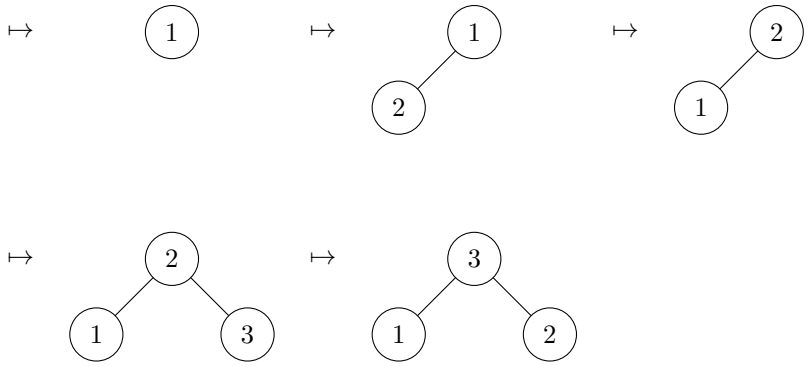
```
Prozedur EINFÜGEN(Q, D)
    Q.HLänge := Q.HLänge + 1          ▷ Annahme: Q.Länge ist hinreichend groß
    Q[Q.HLänge] := D
    AUFWÄRTSKORRIGIEREN(Q, Q.HLänge)
Ende Prozedur

Prozedur AUFWÄRTSKORRIGIEREN(Q, i)
    Solange i > 1 und Q[VATER(i)].x < Q[i].x
        Vertausche Q[i] und Q[VATER(i)]
        i := VATER(i)
    Ende Solange
Ende Prozedur
```

Algorithmus 23 Haldensortieren (engl. *heapsort*)

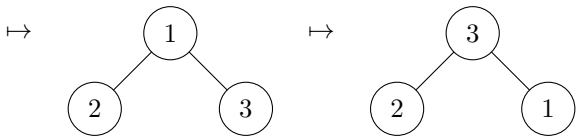
```
Prozedur HALDENSORTIEREN(A)
    ERZEUGEHALDE(A)
    Für i := A.Länge, ..., 2
        Vertausche A[1] und A[i]
        A.HLänge := A.HLänge − 1
        ABWÄRTSKORRIGIEREN(A, 1)
    Ende Für
Ende Prozedur

Prozedur ERZEUGEHALDE(A)
    A.HLänge := A.Länge
    Für i := VATER(A.HLänge), ..., 1
        ABWÄRTSKORRIGIEREN(A, i)
    Ende Für
Ende Prozedur
```

a Die beiden Prozeduren erzeugen nicht immer denselben Heap. Wir betrachten das Gegenbeispiel $A := [1, 2, 3]$.

ERZEUGE-HALDE'(A)



ERZEUGE-HALDE(A)



b Die **for**-Schleife in der Angabe sorgt dafür, dass EUNFÜGEN $(n-1)$-mal ausgeführt wird. Der Einfachheit halber sei dies aber n.

EUNFÜGEN führt eine Aufwärts-Korrektur der aktuellen Halde an dem letzten Knoten im Datenfeld durch. Wenn man die Halde als Binärbaum auffasst, ist das immer der unterste, rechteste Knoten. Aufwärts-Korrektur springt immer von Sohn zu Vater, insgesamt muss also die gesamte Tiefe $\lceil \log_2 i \rceil$ des aktuellen Binärbaums mit $i = 1, \ldots, n$ Knoten raufgesprungen werden.

$$\begin{aligned}\Omega(n \log n) = \frac{n}{2}(\log_2 n - \log_2 2) = \frac{n}{2}\log_2 \frac{n}{2} = \log_2 \left(\frac{n}{2}\right)^{\frac{n}{2}} &\overset{!}{\leq} \log_2 n! = \log_2 \prod_{i=1}^{n} i = \sum_{i=1}^{n} \log_2 i \\ &\leq T(n) := \sum_{i=1}^{n} \lceil \log_2 i \rceil \leq \sum_{i=1}^{n} (\log_2 n + 1) = n \log_2 n + n = \mathcal{O}(n \log n)\end{aligned}$$

Die Ungleichung mit dem „!“ kommt aus der Vorlesung. Mit den beiden Abschätzungen erhalten wir nun für die Laufzeit $T(n) = \Theta(n \log n)$.

□

Aufgabe 45. Wir betrachten „Wechsel-Geld-Problem“ (WGK). In einer Währung existieren (unlimitiert viele) Münzen im Wert von $1 = d_1 < d_2 < \cdots d_k$ Cent. Ziel des WGK ist es, mit möglichst wenigen Münzen einen gegebenen Betrag von n Cent zu wechseln.

a Bestätigen Sie, dass das WGK die optimale Teilstruktur-Eigenschaft erfüllt, also zeigen Sie, dass eine optimale Lösung für n Cent, also $n = \sum_i c_i d_i$ und $\sum_i c_i$ ist minimal, auch eine optimale Lösung für b und $n-b$ darstellt, wenn b ein Anteil von n ist, welcher durch die in der Darstellung von n verwendeten Münzen zustande kommt, also $b = \sum_i c'_i d_i$ mit $0 \leq c'_i \leq c_i$.

b Sei $A(n)$ die Anzahl der Münzen, die zum Wechseln des Betrags n mindestens notwendig sind. Überlegen Sie sich eine Rekursion für $A(n)$.

c Erklären Sie die Funktionsweise des folgenden Algorithmus WECHSEL(d, k, n), welcher für den Wert n und für ein gegebenes Münzen-Array $d = (d_1, \ldots, d_k)$ die optimale Wechsellösung liefert am Beispiel $(d_1, d_2, d_3) = (1, 2, 5)$ und $n = 8$.

```
procedure WECHSEL(d, k, n)
    Seien C[0, ..., n] und S[0, ..., n] neue Felder
    C[0] := 0
    for j = 1 to n do
        C[j] := ∞
        for i = 1 to k do
            if d_i ≤ j and 1 + C[j − d_i] < C[j] then
                C[j] := 1 + C[j − d_i]
                S[j] := d_i
            end if
        end for
    end for
end procedure
```

d Wie groß ist asymptotisch die Komplexität von WECHSEL(d, k, n)?

Lösung.

a Wir werden die Indizes aus unserer Notation entfernen. Sei c eine optimale Lösung für den Wert n, d.h. $n = cd$ und $\sum c$ ist minimal. Sei weiters $0 \leq c' \leq c$ und $c'd = b$. Angenommen, es gäbe $\bar{c}$ mit $\bar{c}d = b$ und $\sum \bar{c} < \sum c'$.

$$\implies (\bar{c} + c - c')d = \bar{c}d + cd - c'd = b + n - b = n, \quad \sum(c + \bar{c} - c') = \sum c + \underbrace{\sum \bar{c} - \sum c'}_{<0} < \sum c$$

$c + \bar{c} - c'$ wäre also echt besser als c. Widerspruch!

Alternative:

Sei $n \in \mathbb{N}$ und für alle $i \in \{1, \ldots, k\}$ sei $c_i \in \mathbb{N}$ mit

$$n = \sum_{i=1}^{k} c_i d_i \quad \text{mit} \quad \sum_{i=1}^{k} c_i \quad \text{minimal.}$$

Nun betracthen wir für jedes $i \in \{1, \ldots, k\}$ ein $c_i' \in \{0, \ldots, c_i\}$ und ein $e_i \in \mathbb{N}$ mit

$$b = \sum_{i=1}^{k} c_i' d_i = \sum_{i=1}^{k} e_i d_i.$$

Wir erkennen

$$n = b + n - b = \sum_{i=1}^{k} (e_i + c_i - c_i')d_i$$

und daher

$$\sum_{i=1}^{k} c_i d_i \leq \sum_{i=1}^{k} (e_i + c_i - c'_i) \Leftrightarrow \sum_{i=1}^{k} c'_i \leq \sum_{i=1}^{k} e_i.$$

Also ist

$$\sum_{i=1}^{k} c'_i \quad \text{minimal.}$$

b Der Rekursions-Anfang ist klar. Für den Rekursions-Schritt betrachten wir ein $i = 1, \ldots, k$ mit $d_i \leq n$. $A(n - d_i)$, die minimale Anzahl der Münzen, um den Wert $n - d_i$ darzustellen, ist bereits bekannt.

Laut a) ist die WGK-Lösung von $n - d_i$ eine Teil-Lösung derer von n. Dieser Fehlt genau 1 Münze im Wert von d_i. Es genügt also, über alle $A(n - d_i) + 1$, für $i = 1, \ldots, k$ mit $d_i \leq n$ zu minimieren, um $A(n)$ zu bekommen.

$$A(0) = 0, \quad A(n) = \min \{A(n - d_i) + 1 : i = 1, \ldots, k \wedge d_i \leq n\}$$

c

1. Lösung:

   Wir bemerken zunächst, dass innerhalb der äußeren **for**-Schleife die Struktur der oberen Formel für $A(n)$ realisiert wird. Um zu minimieren, wird anfangs das größt-mögliche Element ∞ betrachtet und dann, falls letztere Bedingung in der min-Menge gilt, sukzessive verringert. Wir gehen den Algorithmus nun zum Teil durch.

   Außerhalb der Schleifen

	0	1	2	3	4	5	6	7	8
C	0								
S									

   $j = 1$

	0	1	2	3	4	5	6	7	8
C	0	∞							
S									

   $i = 1$
   Nun ist $1 = d_1 \leq 1$ und $1 + 0 = 1 + C[1 - d_1] < C[1] = \infty$, also ...

	0	1	2	3	4	5	6	7	8
C	0	$1 + C[1 - d_1] = 1$							
S		$d_1 = 1$							

   $i = 2$
   Nun ist $2 = d_2 \not\leq 1$.

$\underline{i = 3}$
Nun ist $5 = d_3 \not\leq 1$.

$\underline{j = 2}$

	0	1	2	3	4	5	6	7	8
C	0	1	∞						
S		1							

$\underline{i = 1}$
Nun ist $1 = d_1 \leq 2$ und $1 + 1 = 1 + C[2 - d_1] < C[2] = \infty$.

	0	1	2	3	4	5	6	7	8
C	0	1	$1 + C[2 - d_1] = 2$						
S		1	$d_1 = 1$						

$\underline{i = 2}$
Nun ist $2 = d_2 \leq 2$ und $1 + 0 = 1 + C[2 - d_2] < C[2] = 2$.

$\vdots$

	0	1	2	3	4	5	6	7	8
C	0	1	1	2	2	1	2	2	3
S		1	2	1	2	5	1	2	1

2. Lösung:
Zuerst betrachten wir die optimale Lösung für den Wechsel $j = 1$er Münze und erhalten

$$C = [0, 1]$$
$$S = [*, d_1 = 1],$$

wobei $*$ für einen beliebigen Wert steht. Danach basierend auf dieser Lösung die optimale Lösung für $j = 2$ Münzen. Wir erhalten

$$C = [0, 1, 1]$$
$$S = [*, 1, d_2 = 2].$$

Beim nächsten Schritt für $j = 3$ Münzen wird es interessanter. Hier haben wir prinipiell zwei mögliche vorgangsweisen. Wir könnten $S[3] = d_1$ setzen und als Fortsetzung zu $j = 2$ betrachten oder aber wir setzen $S[3] = d_2$ und betrachten es als Fortsetzung von $j = 1$. In beiden Fällen erreichen wir die gleiche Aufteilung $d_1 + d_2 = 1 + 2 = 3$. Für welchen Wert $S[3]$ entscheidet sich unser Algorithmus? Er nimmt jenen mit kleinerem Index, also

$$C = [0, 1, 1, 2]$$
$$S = [*, 1, 2, d_1 = 1]$$

So geht das weiter. Wie können wir die Münzen welche wir brauchen, herausfinden wenn wir die Ausgaben C und S vom Algorithmus bekommen? Wir nehmen zuerst $S[n]$, danach $S[n - S[n]]$, danach $S[n - S[n] - S[n - S[n]]]$ und so weiter, so lange bis das Argument Null wird.

d Die 1-te **for**-Schleife hat maximal n Durchläufe, die 2-te k. Der asymptotische Aufwand ist daher $\mathcal{O}(nk)$.

□

Aufgabe 46. Das (diskrete) Rucksack-Problem. Sie haben schon wieder gewonnen! Diesmal dürfen Sie sich unter n Gegenständen (die Sie aber nicht zerteilen dürfen), wobei der i-te Gegenstand v_i Euro wert ist und w_i Kilo wiegt, so viele aussuchen und in Ihrem Rucksack hineinpacken, so viel Sie tragen können. Wir nehmen dabei weiters immer an, dass v_i, w_i und W positive ganze Zahlen sind. Die Aufgabe beim „Rucksack-Problem“ besteht nun darin, anzugeben, welche Gegenstände Sie mitnehmen sollen, sodass der Gesamtwert der eingepackten Gegenstände möglichst groß ist.
Etwas genauer: Gesucht ist $f(n, W)$, wenn

$$f(m, W') = \max \left\{ \sum_{i=1}^{m} x_i v_i \;\middle|\; \sum_{i=1}^{m} x_i w_i \leq W' \text{ und } x_i \in \{0, 1\} \right\},$$

für $m \in \{1, 2, \ldots, n\}$ und $W' \in \{0, \ldots, W\}$.

a Überlegen Sie sich, dass das Rucksack-Problem die optimale Teilstruktur-Eigenschaft (wie lautet diese hier?) besitzt.

b Geben Sie eine Rekursion für $f(m, W')$, also für den Wert von optimalen Teillösungen des ursprünglichen Problems an.

Anmerkung: Diese Rekursion hängt nun anders als beim vorigen Beispiel von beiden Parametern m und W' ab.

c Man gebe einen Algorithmus an, der unter Zuhilfenahme von dynamischer Programmierung das Rucksack-Problem löst.

Lösung.

a **Behauptung.** Seien $\sum_{i=1}^{n} x_i v_i = f(n, W)$, d.h. eine optimale Lösung zum Gewicht W und $m \in \{1, \ldots, n\}$. Dann ist $\sum_{i=1}^{m} x_i v_i = f(m, W')$, d.h. eine optimale Lösung zum Gewicht $W' := W - \sum_{i=m+1}^{n} x_i w_i$ und den Gegenständen $1, \ldots, m$. $\sum_{i=m+1}^{n} x_i v_i$ ist dann auch eine optimale Lösung zum Gewicht $\sum_{i=m+1}^{n} x_i w_i$ und den Gegenständen $m + 1, \ldots, n$.

Beweis. Seien unsere $x_1, \ldots, x_n$, sodass $\sum_{i=1}^{n} x_i v_i$ maximal ist, unter der Bedingung $\sum_{i=1}^{n} x_i w_i \leq W$. Angenommen

$$\exists \tilde{x}_1, \ldots, \tilde{x}_m : \sum_{i=1}^{m} \tilde{x}_i v_i > \sum_{i=1}^{m} x_i v_i, \quad \sum_{i=1}^{m} \tilde{x}_i w_i \leq W'.$$

Dann wäre aber

$$\sum_{i=1}^{m} \tilde{x}_i v_i + \sum_{i=m+1}^{n} x_i v_i > \sum_{i=1}^{n} x_i v_i, \quad \sum_{i=1}^{m} \tilde{x}_i w_i + \sum_{i=m+1}^{n} x_i w_i \leq W.$$

Widerspruch!

b Wir verwenden die oben gezeigt Teilstruktur-Eigenschaft für $m = n - 1$.

$$f(\cdot, 0) = f(0, \cdot) = 0, \quad f(n, W) = \begin{cases} f(n-1, W), & w_n > W, \\ \max\{f(n-1, W), v_n + f(n-1, W - w_n)\}, & w_n \leq W \end{cases}$$

c
```
procedure f(n, W)
    Sei f[0,...,n][1,...,W] = 0 ein neues Datenfeld.
    for i = 1,...,n do
        for j = 1,...,W do
            if w_i ≤ j then
                f[i,j] = max{v_i + f[i-1, j-w_i], f[i-1,j]}
            else
                f[i,j] = f[i-1,j]
            end if
        end for
    end for
    return f[n,W]
end procedure
```

□

Aufgabe 47. Nehmen Sie an, Sie wollen die Ausgaben 0 und 1 mit Wahrscheinlichkeit je $\frac{1}{2}$ erhalten. Dazu steht Ihnen die Prozedur Biased-Random zur Verfügung, die den Wert 1 mit Wahrscheinlichkeit p und den Wert 0 mit Wahrscheinlichkeit $1 - p$ ausgibt ($0 < p < 1$). Sie wissen aber nicht, wie groß p ist. Geben Sie einen Algorithmus (in Pseudocode) an, der Biased-Random als Unterroutine verwendet und den Wert 0 mit Wahrscheinlichkeit $\frac{1}{2}$ und den Wert 1 ebenfalls mit Wahrscheinlichkeit $\frac{1}{2}$ zurückgibt. Wie groß ist die erwartete Laufzeit ihres Algorithmus als Funktion von p?

Lösung. Seien X, Y unabhängige Instanzen von Biased-Random().

$$\begin{aligned} \implies \mathbb{P}(X=0, Y=0) &= \mathbb{P}(X=0)\mathbb{P}(Y=0) = p^2, \\ \mathbb{P}(X=0, Y=1) &= \mathbb{P}(X=0)\mathbb{P}(Y=1) = p(1-p) = \\ \mathbb{P}(X=1, Y=0) &= \mathbb{P}(X=1)\mathbb{P}(Y=0) = (1-p)p, \\ \mathbb{P}(X=1, Y=1) &= \mathbb{P}(X=1)\mathbb{P}(Y=1) = (1-p)^2 \end{aligned}$$

Die folgende Prozedur liefert also, mit X bzw. Y, in einer Hälfte der Fälle 0 und in der anderen 1.

```
procedure Random
    X := Y := 0
    while X = Y do
        X = Biased-Random()
        Y = Biased-Random()
    end while
    return X (oder Y)
end procedure
```

$$\begin{aligned}\mathbb{P}(\text{continue}) &:= \mathbb{P}(X = Y)\\ &= \mathbb{P}((X = 0 \wedge Y = 0) \vee (X = 1 \wedge Y = 1))\\ &= \mathbb{P}(X = 0)\mathbb{P}(Y = 0) + \mathbb{P}(X = 1)\mathbb{P}(Y = 1)\\ &= p^2 + (1-p)^2\\ &= p^2 + 1 - 2p + p^2\\ &= 1 - 2p + 2p^2\\ &= 1 - 2p(1-p)\end{aligned}$$

$$\mathbb{P}(\text{break}) := 1 - \mathbb{P}(\text{continue}) = 2p(1-p) =: q \in (0,1)$$

$$\mathbb{P}(\text{Durchläufe} = n) := (1-q)^{n-1}q$$

$$\forall\, x \in (-1,1) : \sum_{n\in\mathbb{N}} nx^{n-1} = \sum_{n\in\mathbb{N}} \frac{\mathrm{d}}{\mathrm{d}x}x^n = \frac{\mathrm{d}}{\mathrm{d}x}\sum_{n\in\mathbb{N}} x^n = \frac{\mathrm{d}}{\mathrm{d}x}\frac{1}{1-x} = \frac{1}{(1-x)^2} \qquad (*)$$

$$\mathbb{E}(\text{Durchläufe}) := \sum_{n\in\mathbb{N}} n\mathbb{P}(\text{Durchläufe} = n) = q\sum_{n\in\mathbb{N}} n(1-q)^{n-1} \overset{(*)}{=} \frac{q}{(1-(1-q))^2} = \frac{q}{q^2} = \frac{1}{2p(1-p)}$$

□

Aufgabe 48. Es gibt zwei Arten von randomisierten Algorithmen: Las-Vegas-Algorithmen nutzen randomisierten Input, um die erwartete Laufzeit zu verringern, produzieren aber immer ein korrektes Ergebnis (man gambelt mit der Laufzeit). Monte-Carlo-Algorithmen hingegen haben deterministische Laufzeit, produzieren aber mit einer gewissen Wahrscheinlichkeit ein falsches oder gar kein Ergebnis (insofern ist die Bezeichung „Algorithmus“ auch etwas fragwürdig).
Betrachten wir nun das Problem, in einem Array A der Größe $2n$ bestehend aus n Einträgen mit Wert a und n Einträgen mit Wert b ein a zu finden, sowie einen Las-Vegas- und einen Monte-Carlo-Algorithmus, der dieses Problem löst:

Las-Vegas-Algorithmus:

```
LV-FIND-a(A)
while true do
    i = RANDOM(1, 2n)
    if A[i] = a then
        return i
    end if
end while
```

Dieser Algorithmus liefert immer ein richtiges Ergebnis. Bestimmen Sie seine erwartete Laufzeit.

Monte-Carlo-Algorithmus:

```
MC-FIND-a(A, k)
```

$i = 0$
while $i < k$ **do**
  $i = \text{RANDOM}(1, 2n)$
  if $A[i] = a$ **then**
    return i
  end if
end while

Dieser Algorithmus braucht höchstens k Schritte (k fest), hat also konstante Laufzeit. Bestimmen Sie die Wahrscheinlichkeit, dass nach k Durchläufen ein a gefunden wurde.

Lösung.

$$\forall\, i = 1, \ldots, 2n : \mathbb{P}(A[i] = a) = \mathbb{P}(A[i] = b) = \frac{1}{2}$$
$$\implies \mathbb{P}(A[\text{RANDOM}(1, 2n)] = a) = \mathbb{P}(A[\text{RANDOM}(1, 2n)] = b) = \frac{1}{2}$$

1. Algorithmus (Las-Vegas):

$$\forall\, k \in \mathbb{N} : \mathbb{P}(\text{Durchläufe} = k) = \mathbb{P}(A[\text{RANDOM}(1, 2n)] = b)^{k-1}\mathbb{P}(A[\text{RANDOM}(1, 2n)] = a)$$
$$= \left(\frac{1}{2}\right)^{k-1}\frac{1}{2}$$

$$\mathbb{E}(\text{Durchläufe}) = \sum_{k\in\mathbb{N}} k\mathbb{P}(\text{Durchläufe} = k) = \sum_{k\in\mathbb{N}} n\left(\frac{1}{2}\right)^{k-1}\frac{1}{2} = \frac{1}{2}\sum_{k\in\mathbb{N}} n\left(\frac{1}{2}\right)^{k-1} \overset{(*)}{=} \frac{1}{2}\frac{1}{\left(1-\frac{1}{2}\right)^2} = 2$$

2. Algorithmus (Monte-Carlo):

$$\mathbb{P}(a \in A[1:k]) = 1 - \mathbb{P}(a \notin A[1:k]) = 1 - \mathbb{P}(A[1] = b, \ldots, A[k] = b) = 1 - \left(\frac{1}{2}\right)^k$$

□

Diskrete und Geometrische Algorithmen

9. Übung

Richard Weiss Florian Schager Christian Sallinger Fabian Zehetgruber
Paul Winkler Christian Göth

18.01.2021

Aufgabe 49.

a) Erzeugt die folgende Modifikation des Fisher-Yates-Algorithmus ebenfalls eine zufällige Permutation von A? Weshalb oder weshalb nicht?

PERMUTE-WITH-ALL($A[1,\dots,n]$)
$n = A$.length
for $i = 1$ to n **do**
  Vertausche $A[i]$ mit $A[\text{RANDOM}(1,n)]$
end for

b) Eine weitere Abwandlung des Fisher-Yates-Algorithmus: Diesmal wird das Element $A[i]$ in jedem Schritt mit einem zufälligen Element aus dem Teilfeld $A[i+1,\dots,n]$ vertauscht, also

PERMUTE-WITHOUT-IDENTITY($A[1,\dots,n]$)
$n = A$.length
for $i = 1$ to n **do**
  Vertausche $A[i]$ mit $A[\text{RANDOM}(i+1,n)]$
end for

Man könnte zunächst meinen, dass dieser Algorithmus alle von der Identität verschiedenen Permutationen zufällig erzeugt. Weisen Sie nach, dass dies aber nicht der Fall ist. Überlegen Sie sich, welche Klasse von Permutationen von diesem Algorithmus tatsächlich zufällig erzeugt werden.

Lösung.

a) Der Algorithmus kann im Allgemeinen keine zufällige Permutation von A liefern. Das sieht man daran, dass in jedem Schleifendurchlauf n gleich wahrscheinliche Resultate auftreten. Nach dem n-ten Schleifendurchlauf erhalten wir somit genau n^n gleich wahrscheinliche (nicht notwendigerweise verschiedene) Permutationen von A. Bereits für $n = 3$ kann dabei nicht jede Permutation gleich häufig auftreten da $\frac{3^3}{3!} = \frac{27}{6}$ keine ganze Zahl ist.

b) Der Algorithmus PERMUTE-WITHOUT-IDENTITY kann nur Permutationen erzeugen, in denen kein Element von A an der selben Stelle wie am Anfang stehen bleibt.
Bezeichne mit A_i das Datenfeld nach dem i-ten Durchlauf der Schleife und gelte weiters für das Anfangsdatenfeld A_0 o.B.d.A. $A_0[i] = i$ für $i = 1,\dots,n$.
Wir zeigen also mit Induktion nach $i = 1,\dots,n : \forall j \le i : A_i[j] \neq j$.
Der Induktionsanfang $i = 1$ ist klar: $A_1[1] = A[\text{RANDOM}(2,n)] \neq 1$.
Im $(i+1)$-ten Schleifendurchlauf werden die ersten i Elemente des Datenfelds A_i nicht mehr angerührt, also gilt laut Induktionsvoraussetzung $A_{i+1}[j] \neq j$ für $j = 1,\dots,n$. Weiters gilt $A_{i+1}[i+1] = A[\text{RANDOM}(i+2,n)] \neq i+1$.
Jetzt könnte man sich noch überlegen, ob alle Permutationen, die obige Bedingung erfüllen, tatsächlich

durch den Algorithmus erreicht werden können.
Dem ist aber nicht so, wie man an folgendem Beispiel sieht:

$$(1,2,3,4) \rightsquigarrow (4,2,3,1) \rightsquigarrow (4,3,2,1) \rightsquigarrow (4,3,1,2)$$

Also kann die fixpunktfreie Permutation $(4,3,2,1)$ mittels diesem Algorithmus nicht erreicht werden.

□

Aufgabe 50. Bestimmen Sie eine Rekursion für die mittlere Anzahl der Schlüsselvergleiche beim (nicht-randomisierten) Quicksort, wenn anstelle des letzten Elements $A[n]$ da der Größe nach mittlere der drei Elemente $A[1], A[\lfloor \frac{n}{2} \rfloor]$ und $A[n]$ als Pivot-Element verwendet wird. Das Lösen dieser Rekursion ist nicht verlangt.

Lösung. Wir berechnen zuerst die Wahrscheinlichkeit, dass das mittlere von drei zufällig gewählten Elementen aus $[1,\dots,n]$ gleich j ist.
Die Anzahl aller Kombinationen von 3 aus n Elementen ohne Wiederholung ist genau $\binom{n}{3}$. Die Wahrscheinlichkeit dass das mittlere Element der Kombination gleich j ist, lässt sich umformulieren als die Wahrscheinlichkeit, dass j in der Kombination liegt und zusätzlich genau ein Element aus $[1,\dots,j-1]$ ebenfalls in der Kombination liegt. Seien (k_1,k_2,k_3) die aufsteigend sortierten Elemente der Kombination k. Für $2 \le j \le n-1$ gilt dann

$$\mathbb{P}(k_2 = j) = \mathbb{P}(j \in k \text{ und } |[1,\dots,j-1] \cap k| = 1) = \frac{3}{n}\left[\left(1-\frac{2}{n-1}\right)^{j-2}\frac{2}{n-1}(j-1)\right] = (j-1)\left(\frac{2}{n-1}\right)^2\left(1-\frac{2}{n-1}\right)^{j-2}$$

Damit erhalten wir unsere Rekursion

$$T(n) = \sum_{j=2}^{n-1}(j-1)\left(\frac{2}{n-1}\right)^2\left(1-\frac{2}{n-1}\right)^{j-2}[T(n-j)+T(j-1)] + n$$

□

Lösung. Das Ergebins hier schaut anders aus, ist es womöglich äquivalent? Sei $X_n : S_n \to \{1,\dots,n\}$ eine Zufallsvariable, welche jede Permutation A der Elemente $1,\dots,n$ abbildet auf das mittlere der drei $A[1], A[\lfloor \frac{n}{2} \rfloor]$ und $A[n]$. Für jedes $1 < j < n$ gilt

$$\begin{aligned}\mathbb{P}(X_n = j) &= 6\mathbb{P}\Big(A[1] < A\Big[\Big\lfloor\frac{n}{2}\Big\rfloor\Big] \wedge A\Big[\Big\lfloor\frac{n}{2}\Big\rfloor\Big] < A[n] \wedge A\Big[\Big\lfloor\frac{n}{2}\Big\rfloor\Big] = j\Big)\\ &= 6\mathbb{P}\Big(A\Big[\Big\lfloor\frac{n}{2}\Big\rfloor\Big] = j\Big)\mathbb{P}(A[1] < j)\mathbb{P}(j < A[n]) = \frac{6}{n}\frac{j-1}{n-1}\frac{n-j}{n-2}.\end{aligned}$$

Wir erhalten für die Laufzeit also die Rekursion

$$T(n) = \mathcal{O}(n) + \sum_{j=2}^{n-1}\frac{6}{n}\frac{j-1}{n-1}\frac{n-j}{n-2}[T(n-j)+T(j-1)].$$

Bezeichne $S(n)$ die durchschnittliche Anzahl der Schlüsselvergleiche, dann erhalten wir die Rekursion

$$S(n) = \sum_{j=2}^{n-1}\frac{6}{n}\frac{j-1}{n-1}\frac{n-j}{n-2}[S(n-j)+S(j-1)] + 3$$

mit $S(0) = S(1) = 0$ und $S(2) = 1$. □

Aufgabe 51. Quickselect ist ein Algorithmus zum Auffinden des j-kleinsten Elements eines Feldes, der auf einer ähnlichen Idee wie Quicksort basiert: Wähle ein zufälliges Pivotelement und bringe es wie bei Quicksort an die richtige Stelle, sodass im Teilfeld links davon nur kleinere Elemente und im Teilfeld rechts davon nur größere Elemente sind. Ist das Pivotelement genau an Stelle j, so sind wir fertig. Hat das linke Teilfeld mehr als j Elemente, so suche dort weiter, ansonsten im rechten Teilfeld (anstatt wie bei Quicksort beide Seiten rekursiv zu betrachten, macht man dies jetzt nur mit der Seite, wo das Element ist).
Schreiben Sie einen Pseudocode für den Algorithmus Quickselect.

Lösung.

```
1: QUICKSELECT(A, j, l, r)
2: if l < r then
3:     m := TEILEN(A, l, r)
4:     if m > j then
5:         QUICKSELECT(A, j, l, m − 1)
6:     else if m < j then
7:         QUICKSELECT(A, j, m + 1, r)
8:     else if m = j then
9:         return m
10:    end if
11: end if
```

```
1: TEILEN(A, l, r)
2: x := A[r]
3: i := l
4: for j = l, . . . , r − 1 do
5:     if A[j] ≤ x then
6:         Vertausche A[i] mit A[j]
7:         i := i + 1
8:     end if
9: end for
10: Vertausche A[i] mit A[r]
11: return i
```

□

Aufgabe 52. Zur Analyse der erwarteten Laufzeit von Quickselect: Bezeichne $T(n)$ die Laufzeit von Quickselect bei einem Feld der Größe n und sei X_k das Ereignis, dass das Pivotelement an k-ter Stelle steht.

a) Weisen Sie nach, dass $\mathbb{E}(X_k) = \frac{1}{n}$ für $k = 1, \ldots, n$ und dass für $T(n)$ gilt, dass

$$T(n) \leq \sum_{k=1}^{n} X_k \cdot T(\max(k-1, n-k)) + \mathcal{O}(n).$$

b) Da X_k und $T(\max(k-1, n-k))$ unabhängig sind, lässt sich also folgern, dass

$$\mathbb{E}(T(n)) \leq \sum_{k=1}^{n} \frac{1}{n} \cdot \mathbb{E}(T(\max(k-1, n-k))) + \mathcal{O}(n).$$

Zeigen Sie (mittels Betrachtung von $\max(k-1, n-k)$), dass

$$\mathbb{E}(T(n)) \leq \frac{2}{n} \sum_{k=\lceil n/2 \rceil}^{n} \mathbb{E}(T(k)) + \mathcal{O}(n).$$

c) Folgern Sie daraus nun mittels Substitutionsmethode, dass $\mathbb{E}(T(n)) = \mathcal{O}(n)$ (also $\mathbb{E}(T(n)) \leq cn$ für passendes $c > 0$).

Lösung.

a) Wir verwenden das Permutationsmodell und erhalten $\mathbb{E}(X_k) = \mathbb{P}(X_k) = \frac{1}{n}$. Wir erhalten weiters

$$T(n) = \sum_{k=1}^{j-1} X_k T(n-k) + \sum_{k=j+1}^{n} X_k T(k-1) + \mathcal{O}(n) \leq \sum_{k=1}^{n} X_k \cdot T(\max(k-1, n-k)) + \mathcal{O}(n)$$

b)

$$\begin{aligned}
\mathbb{E}(T(n)) &\leq \sum_{k=1}^{n} \frac{1}{n} \cdot \mathbb{E}(T(\max(k-1, n-k))) + \mathcal{O}(n) \\
&= \sum_{k=1}^{\lceil n/2 \rceil} \frac{1}{n} \cdot \mathbb{E}(T(n-k)) + \sum_{k=\lceil n/2 \rceil + 1}^{n} \frac{1}{n} \cdot \mathbb{E}(T(k-1)) + \mathcal{O}(n) \\
&= \sum_{k=n-\lceil n/2 \rceil}^{n-1} \frac{1}{n} \cdot \mathbb{E}(T(k)) + \sum_{k=\lceil n/2 \rceil}^{n-1} \frac{1}{n} \cdot \mathbb{E}(T(k)) + \mathcal{O}(n) \\
&\leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} \mathbb{E}(T(k)) + \mathcal{O}(n).
\end{aligned}$$

c) Vermutung: $\mathbb{E}(T(n)) \leq n$. Zur Vereinfachung der Rechnung vernachlässigen wir das $+\mathcal{O}(n)$.

$$\mathbb{E}(T(n)) \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} k \leq \frac{2}{n} \frac{(n-1)n}{2} = n - 1 < n.$$

□

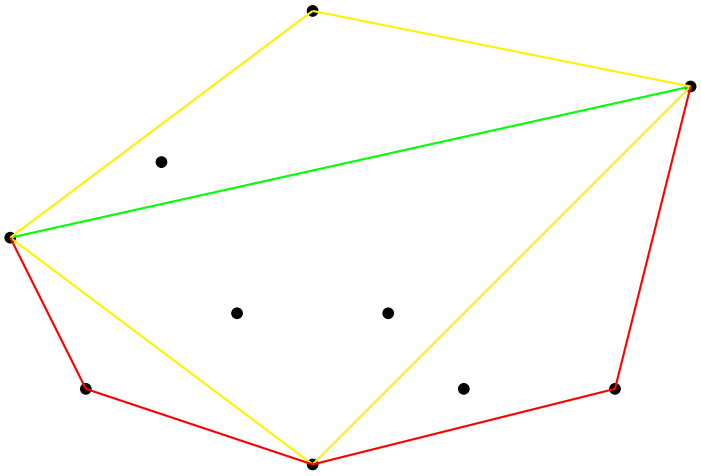
Aufgabe 53. Quick-Hull ist ein auf einer ähnlichen Idee wie Quicksort basierender Divide-And-Conquer-Algorithmus zum Auffinden einer endlichen Punktemenge im $\mathbb{R}^2$. Er arbeitet wie folgt: Zunächst werden die beiden Punkte mit der größten und der kleinsten x-Koordinate gesucht (sollte es mehrere Punkte mit kleinster/größter x-Koordinate geben, so wähle denjenigen mit kleinster y-Koordinate). Da diese Punkte Extremalpunkte sind, sind sie Bestandteil der konvexen Hülle.
Die beiden gefundenen Punkte bilden eine Gerade, die die Punktemenge in zwei Teilmengen unterteilt: Die Punkte rechts von der Gerade und die Punkte links davon (links und rechts ergeben sich aus dem Winkel zwischen dem Richtungsvektor der Geraden und dem Vektor zwischen Anfangspunkt der Gerade und dem betrachtetem Punkt). Diese beiden durch die Gerade getrennten Punktmengen werden nun von Quick-Hull rekursiv betrachtet. Innerhalb der zu betrachtenden Punktmenge wird der Punkt P gesucht, der die maximale Distanz zur Geraden hat. Dieser ist ebenfalls Teil der konvexen Hülle. Das Dreieck bestehend aus den Endpunkten der Gerade und dem Punkt P besteht aus drei Punkten, die alle zur konvexen Hülle gehören. Also können wir alle Punkte im Inneren des Dreiecks bei weiteren Aufrufen des Algorithmus ignorieren.
Die Seiten des Dreiecks fungieren nun als neue Trenngeraden. Der Algorithmus wird so lange wiederholt, bis nur noch die Endpunkte der Trenngerade Teil der zu untersuchenden Punktemengen sind.

a) Illustrieren Sie die Arbeitsweise des Algorithmus anhand einer selbst gewählten Punktemenge mit mindestens 10 Punkten.

b) Begründen Sie, warum $\mathcal{O}(n^2)$ die Worst-Case-Laufzeit von Quick-Hull ist.

Lösung.

a) Lies: Schritt 1 in grün, Schritt 2 in gelb, Schritt 3 in rot.



b) Mit jedem Dreieck, dass wir konstruieren, gewinnen wir zumindest einen Punkt zu unser konvexen Hülle dazu. Also sind wir spätestens nach $n-2$ Dreiecken fertig. Für jedes Dreieck müssen wir schlimmsten Fall die maximale Distanz zur Dreieckskante unter $n-2$ Punkten bestimmen. Also kann unser Aufwand nicht schlimmer als $\mathcal{O}(n^2)$ werden.

□

Aufgabe 54. Wiederholen Sie an Hand des folgenden linearen Programms, wie allgemeine lineare Programme in Standardform gebracht werden können.

Minimiere $2x_1 + 7x_2 + x_3$ unter den Nebenbedingungen
$x_1 - x_3 = 7$,
$3x_1 + x_2 \geq 24$,
$x_2 \geq 0$,
$x_3 \leq 0$.

Lösung. Maximiere $-2(x_1^+ - x_1^-) - 7x_2 + x_3'$ unter den Nebenbedingungen

$$\begin{aligned} -x_1^+ + x_1^- - x_3' &\leq -7 \\ x_1^+ - x_1^- + x_3' &\leq 7 \\ -3(x_1^+ - x_1^-) - x_2 &\leq -24 \\ x_1^+, x_1^-, x_2, x_3' &\geq 0. \end{aligned}$$

Oder in Matrixschreibweise

$$A := \begin{pmatrix} -1 & 1 & 0 & -1 \\ 1 & -1 & 0 & 1 \\ -3 & 3 & -1 & 0 \end{pmatrix} \begin{pmatrix} x_1^+ \\ x_1^- \\ x_2 \\ x_3' \end{pmatrix} \leq \begin{pmatrix} -7 \\ 7 \\ -24 \end{pmatrix} =: b$$

$$\overline{x} \geq 0$$

$$f(x) = \begin{pmatrix} -2 & 2 & -7 & 1 \end{pmatrix} \begin{pmatrix} x_1^+ \\ x_1^- \\ x_2 \\ x_3' \end{pmatrix} \quad \text{max!}$$

□