Tutorium Algorithmen 1

11 · MST, Union Find · 8.7.2024

Peter Bohner Tutorium 3

Motivation MST

Karlsruher Institut für Technologie

Problem:

- Ich möchte mehrere Dinge in meinem Zimmer mit einem Kabel verbinden
  - Benutze Super-Kabel dass alles kann und bel. viele Ausgänge hat
- Super-Kabel sind teuer $\Rightarrow$Möglichst wenig verwenden
- Mehrere Möglichkeiten das Kabel zu verlegen ohne darüber zu stolpern
  - Unterm Bett
  - Unterm Schrank
  - ...

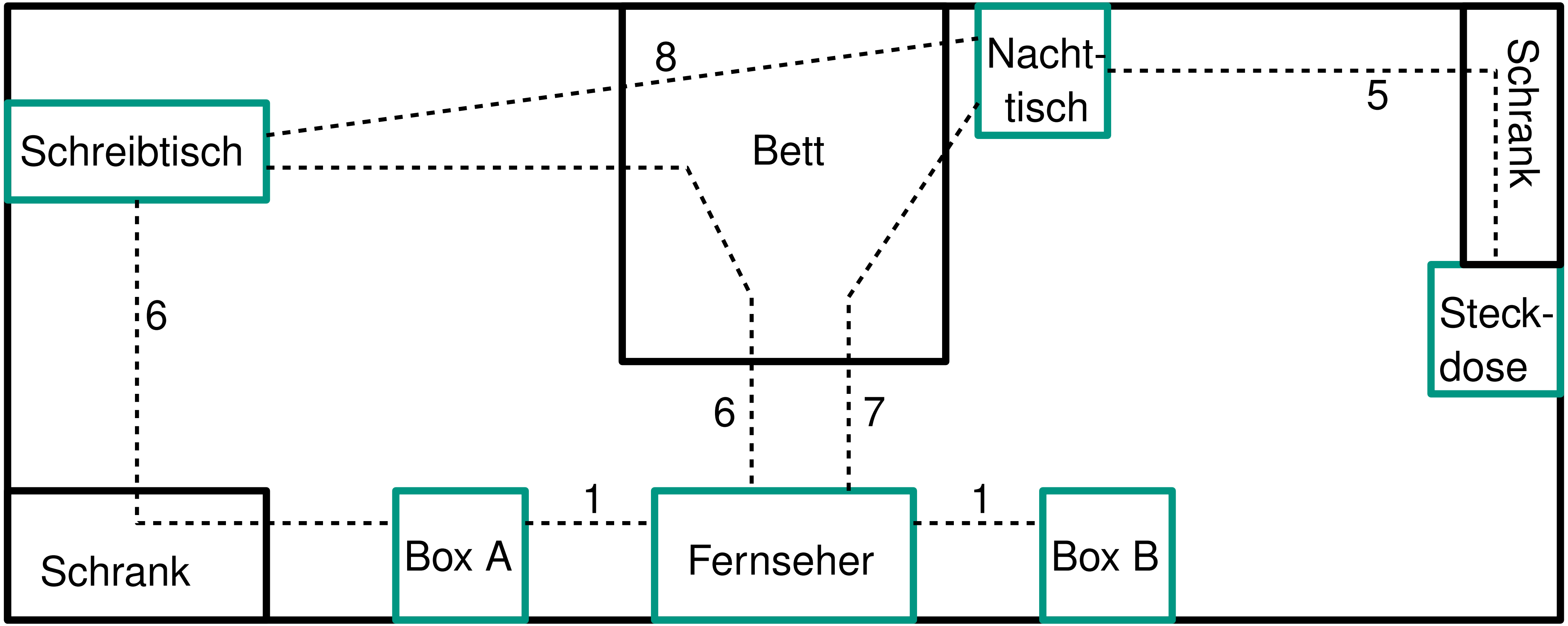
Peter Bohner - peter.bohner@student.kit.edu

Motivation MST

Problem:

- Ich möchte mehrere Dinge in meinem Zimmer mit einem Kabel verbinden

Grober Lageplan:

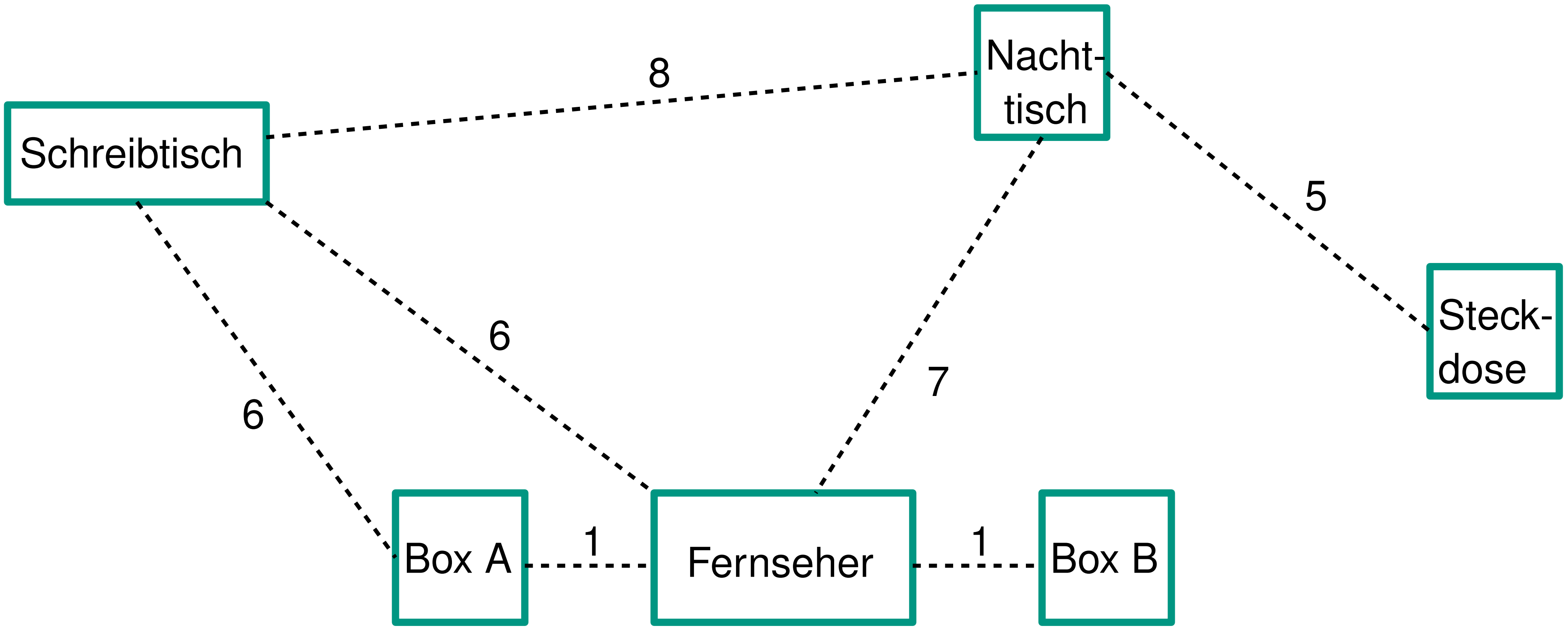


Motivation MST

Problem:

- Ich möchte mehrere Dinge in meinem Zimmer mit einem Kabel verbinden

Etwas abstrahierter:

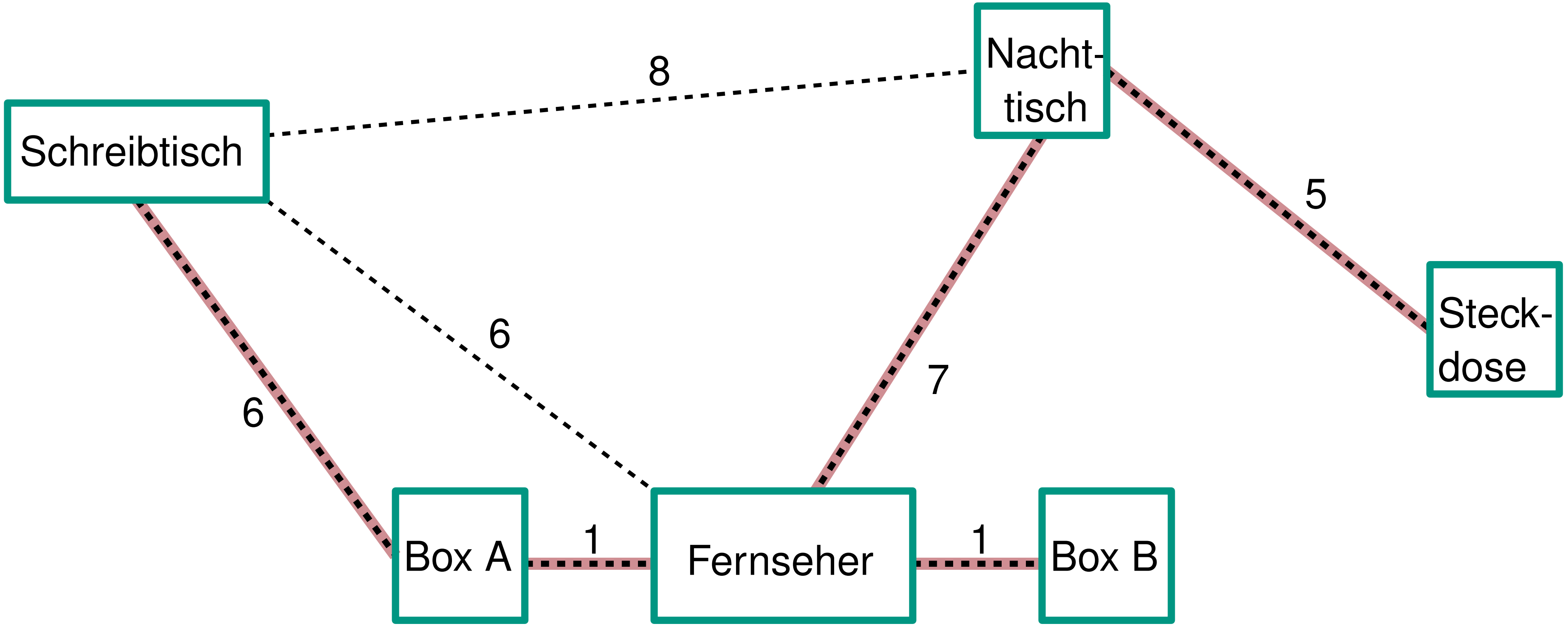


Motivation MST

Problem:

- Ich möchte mehrere Dinge in meinem Zimmer mit einem Kabel verbinden

Optimale Lösung mit Gewicht 20:



MST

- Das war ein **Minimaler Spannbaum (MST)**

Definition
Sei $G = (V, E)$ ein (zusammenhängender) Graph. Ein Baum auf der selben Knotenmenge $T = (V, E_T)$ mit $E_T \subseteq E$ heißt **Spannbaum** von G.

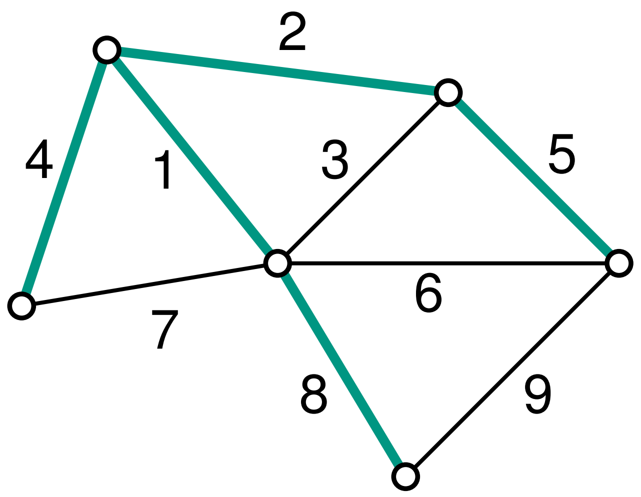
Problem: Minimaler Spannbaum (MST)
Sei $G = (V, E)$ ein Graph mit Kantengewichten $w\colon E \to \mathbb{Z}$. Finde einen Spannbaum $T = (V, E_T)$, sodass die Summe der Gewichte in E_T minimal ist.

Der MST eines Graphen verbindet also alle Knoten mit minimalem Kantengewicht

Peter Bohner - peter.bohner@student.kit.edu

Aufgabe

Findet einen MST auf diesem Graphen



Quiz

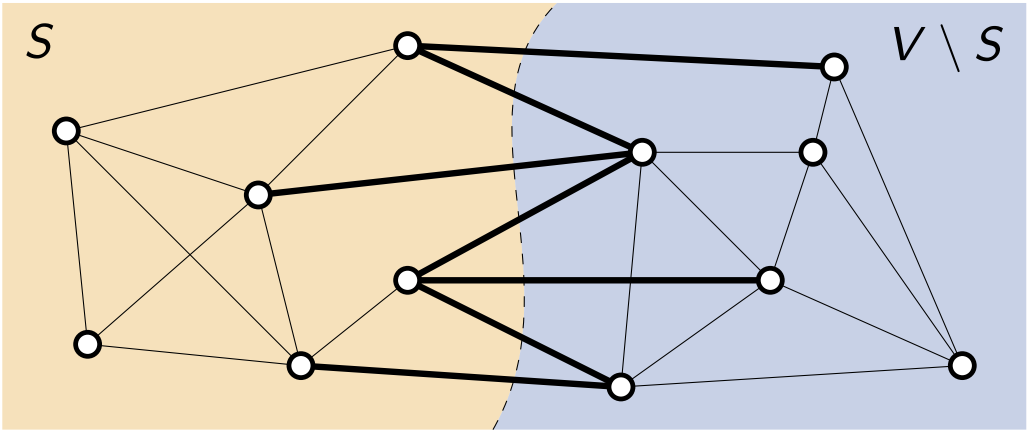
- Welche Kante ist auf jeden Fall in einem MST?
  die leichteste (falls existent)
- Gibt es Graphen, bei denen die schwerste Kante im MST ist? ✓ wenn sie Brücke ist
- Kann die schwerste Kante auf einem Kreis in einem MST enthalten sein? ✗ Austauschargument
- Ist der MST immer eindeutig? ✗
- Welche Voraussetzung ist hinreichend für mehrere MSTs?
  Ein Schnitt hat mehrere minimale Schnittkanten

Peter Bohner - peter.bohner@student.kit.edu

Schnitte

Definition
Sei $G = (V, E)$ ein Graph. Ein **Schnitt** ist eine Zerlegung von V in zwei nicht-leere Teilmengen S und $V \setminus S$. Eine Kante zwischen einem Knoten aus S und einem aus $V \setminus S$ heißt **Schnittkante**.

Das nennt man auch **Schnitteigenschaft**



- Für jeden Schnitt gilt, dass die minimale Schnittkante in jedem MST enthalten ist.
- Beobachtung: Bäume haben $n - 1$ Kanten
- Wenn wir $n-1$ Schnitte finden deren minimale Schnittkante unterschiedlich ist haben wir einen MST
- Genau das macht der Algorithmus von Prim

Algorithmus von Prim

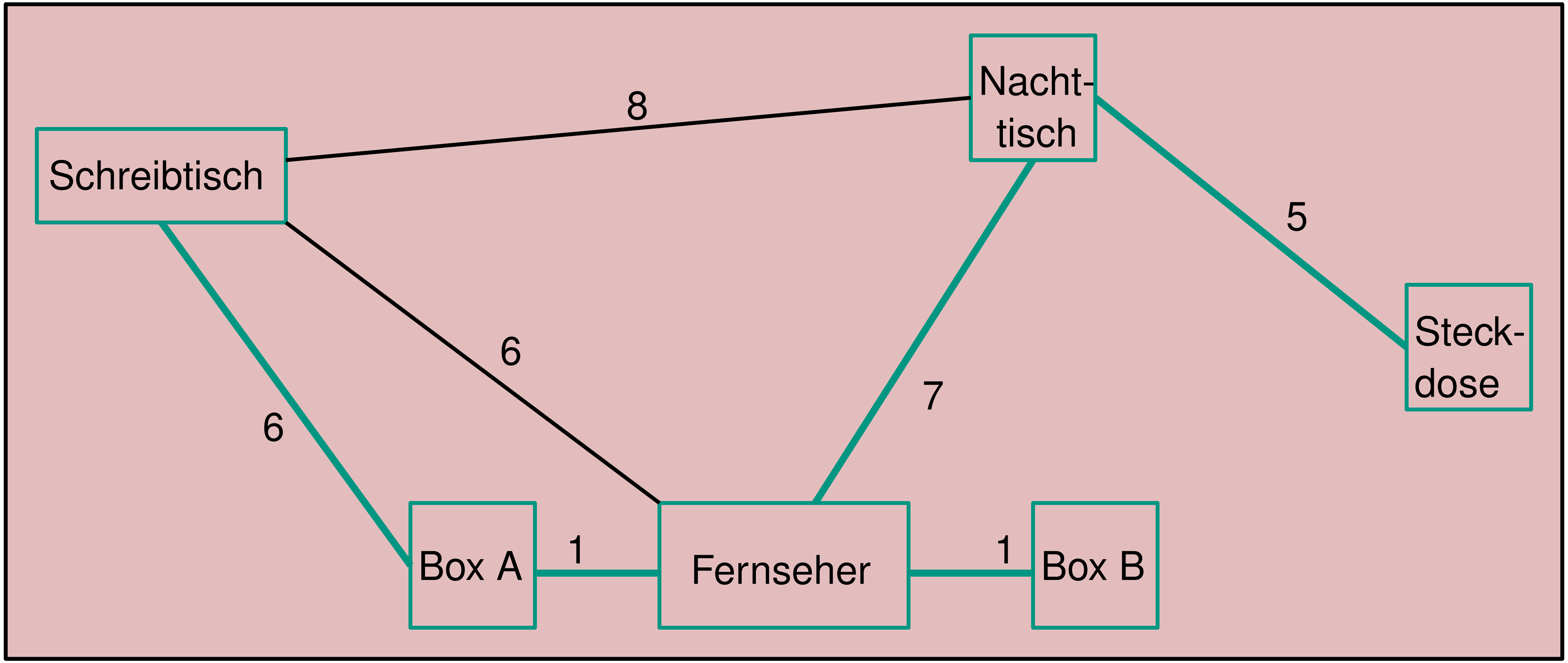
- Wir wollen einen zusammenhängenden Baum bauen
- Wir starten irgendwo und bauen in jedem Schritt einen weiteren Knoten an unseren bisherigen Baum dran.
- Betrachte Schnitt zwischen dem bisherigen Baum und übrigen Knoten, verwende Schnitteigenschaft
- Leichteste von Baum weg gehende Kante ist in einem MST
  $\Rightarrow$füge diese zum Baum hinzu

Umsetzung:

- Wie bei Dijkstra, nur mit leichtestem eingehenden Kantengewicht als Priorität
- Laufzeit: Wie bei Dijkstra als $\mathcal{O}((n+m)\log n)$ bzw. $\mathcal{O}(n+(m\log n))$

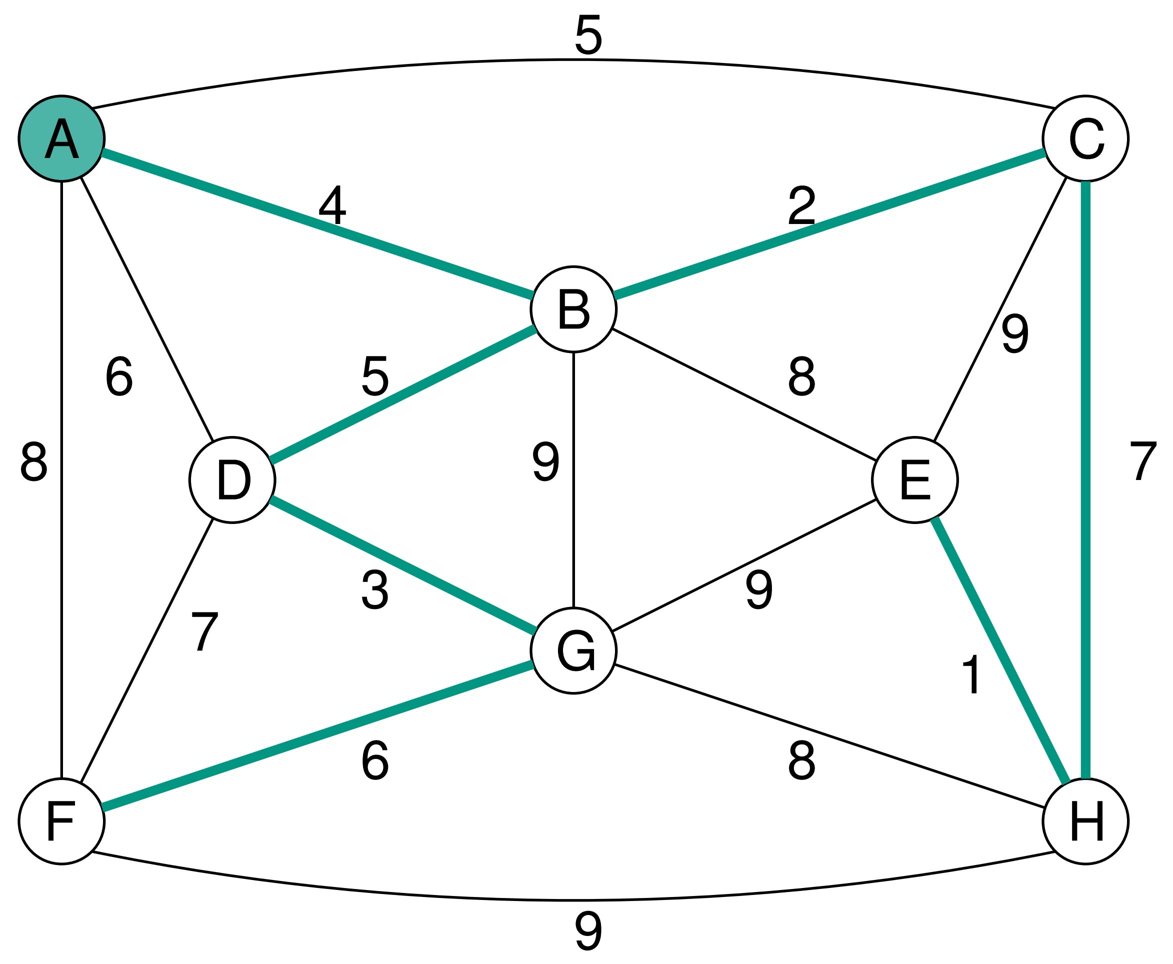
Peter Bohner - peter.bohner@student.kit.edu

Beispiel



Schreibtisch
8
Nacht-
tisch
5
Steck-
dose
6
6
7
Box A
1
Fernseher
1
Box B

Aufgabe



Berechnet einen minimalen Spannbaum mit Prim. Startet dafür in Knoten A

Gebt die Kanten in der Reihenfolge an, in der sie von Prim gefunden wurden

(A,B)
(B,C)
(B,D)
(D,G)
(G,F)
(C,H)
(H,E)

Kreise

Die schwerste Kante auf einem Kreis ist nie Teil eines MST (**Kreiseigenschaft**)

- Angenommen MST T benutzt die schwerste Kante e auf Kreis C
- Dann ex. $e' \in C \backslash T$, so dass $T' = (T \backslash \{e\}) \cup \{e'\}$ auch ein Spannbaum ist
- Es gilt aber $c(e') < c(e) \Rightarrow$T kann kein MST gewesen sein

(Durch Austauschen der Kanten bleiben immer noch alle Knoten auf dem Kreis erreichbar)

> Wir nehmen hier an, dass alle Kantengewichte maximal einmal auftreten

Algorithmus von Kruskal

Idee

- Wir wollen die Summe der gewählten Kanten minimieren.
  $\Rightarrow$ Wähle immer Kante mit geringstem Gewicht
- Soll ein Baum werden.
  $\Rightarrow$ Wähle Kante nur, wenn Kante keinen Kreis schließt
  - Kante die Kreis schließt wäre die größte Kante auf dem Kreis und ist somit nicht im MST enthalten

Alternativ:

- Kante mit geringstem Gewicht, die keinen Kreis schließt, ist leichteste Kante eines Schnitts $\Rightarrow$ Schnitteigenschaft

Peter Bohner - peter.bohner@student.kit.edu

Algorithmus von Kruskal

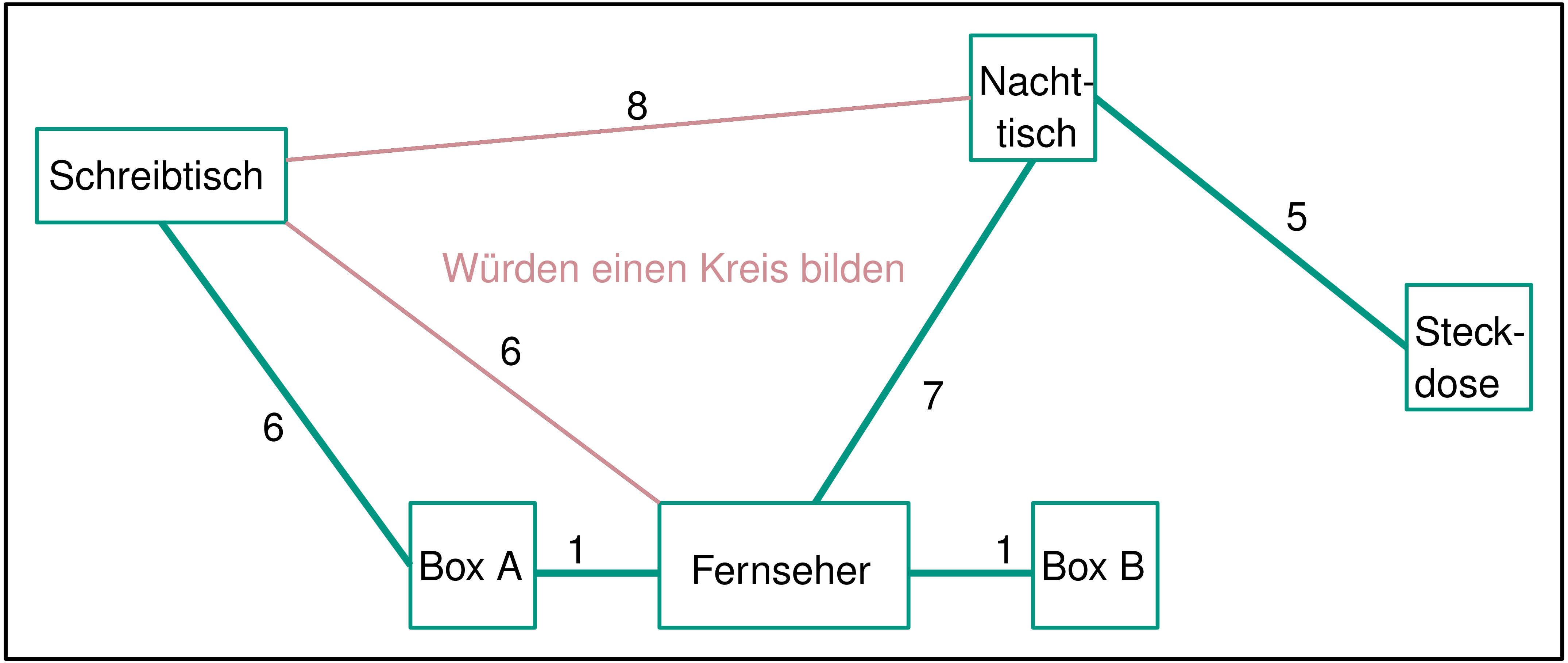
Implementierung

- Gehe Kanten aufsteigend nach Gewicht sortiert durch und wähle alle, die keinen Kreis bilden
- Wie auf Kreise überprüfen?
  - Jedes mal DFS/BFS ist teuer
  - Gleich neue Datenstruktur mit der das schnell geht
- Laufzeit:
  - Sortieren: $\mathcal{O}(m \log m)$
  - Kanten wählen: $\mathcal{O}(m \log^* n)$ (oder $\mathcal{O}(m \cdot \alpha(n))$)

> $\log^* n$ ist der iterierte Logarithmus, in der Realität ist der meistens 4 oder 5, in der Theorie aber nicht Konstant
>
> $\alpha(n)$ ist die inverse Ackermannfunktion (wächst noch langsamer)

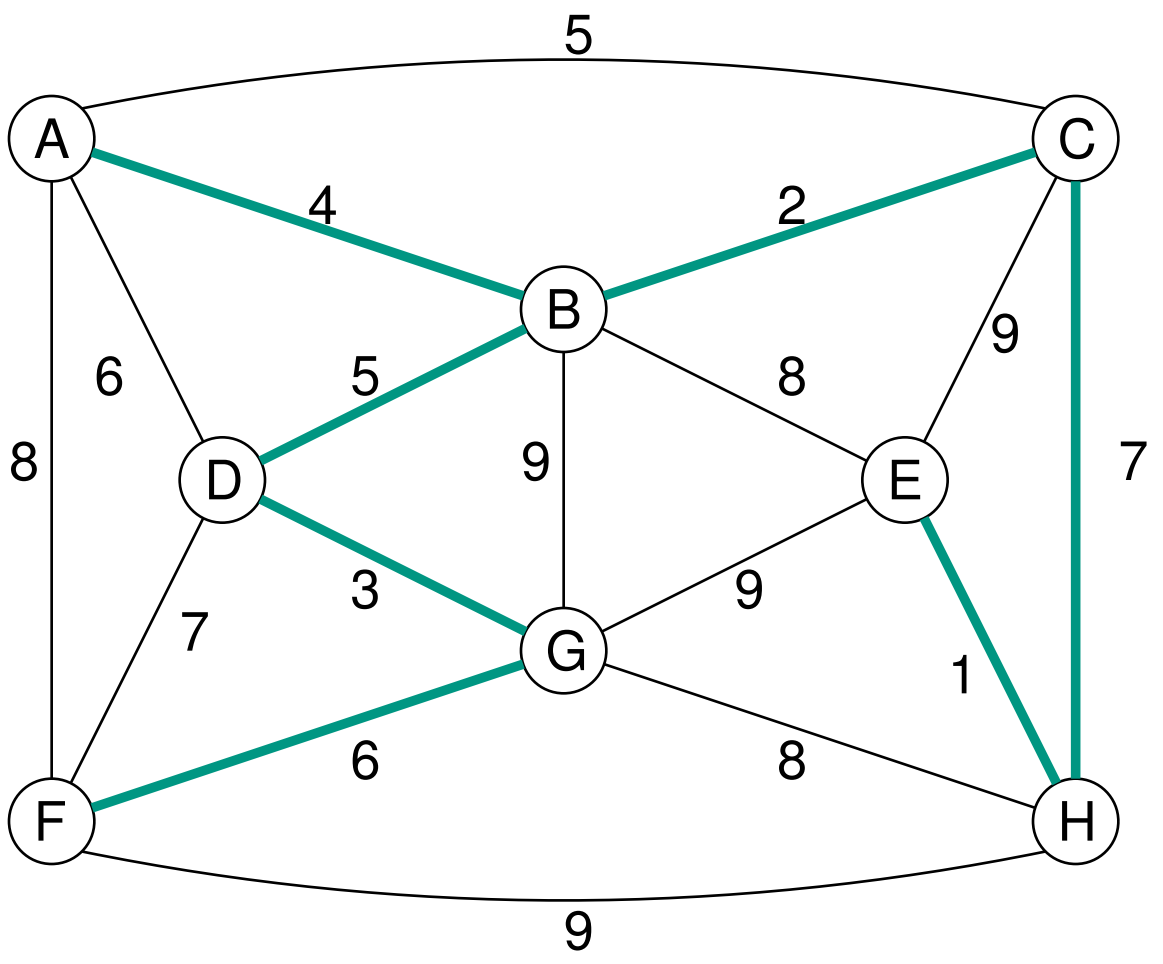
Peter Bohner - peter.bohner@student.kit.edu

Beispiel



Peter Bohner - peter.bohner@student.kit.edu

Aufgabe



Berechnet einen minimalen Spannbaum mit Kruskal.

Gebt die Kanten in der Reihenfolge an, in der sie von Kruskal gefunden wurden

(E,H)
(B,C)
(D,G)
(A,B)
(B,D)
(F,G)
(C,H)

Noch mehr Algorithmen

Berechnen die folgenden Algorithmen einen MST?

1. - T ist zu Beginn leer
   - Kanten werden in beliebiger Reihenfolge betrachtet. Für jede Kante $e \in E$:
   - Falls $T \cup \{e\}$ kreisfrei ist: füge e zu T hinzu
2. - Zu Beginn: $T = G$
   - Kanten werden in absteigender Reihenfolge betrachtet. Für jede Kante $e \in E$:
   - Falls $T \setminus \{e\}$ zusammenhängend ist: lösche e aus T

Peter Bohner - peter.bohner@student.kit.edu

Lösung

1. Nein, da durch die beliebige Reihenfolge auch zu schwerere Kanten in T sein können
   Es gibt sogar für jeden Spannbaum von G eine Reihenfolge, sodass dieser durch den Algorithmus berechnet wird
2. Ja: Kruskal, aber umgekehrt
   - Umgekehrte Reihenfolge der Kanten
   - Ist T nach Entfernen einer Kante immer noch zusammenhängend, so hätte diese in umgekehrter Reihenfolge einen Kreis geschlossen, und umgekehrt

Peter Bohner - peter.bohner@student.kit.edu

Aufgabe

Karlsruher Institut für Technologie

Die Biber haben eine Stadt, bestehend aus n Biberburgen $(B_1, \dots, B_n)$, in der Wüste gebaut. Leider haben sie nicht bedacht, dass sie dort im Trockenen sitzen.
Wasser muss her, aber schnell! Und zwar zu jeder Biberburg. Dazu können die Biber bei beliebigen Biberburgen Brunnen bohren. Die benötigte Zeit t_i für einen Brunnen bei Biberburg B_i hängt von der Beschaffenheit des Bodens dort ab.
Außerdem gibt es m Streifen mit wenig Geröll $(S_1, \dots, S_m)$, die zwischen je zwei Burgen verlaufen und wo die Biber Kanäle bauen können. Die Dauer k_i für den Bau eines Kanals hängt ebenfalls vom Gelände von S_i ab. Wie können die Biber möglichst schnell eine Wasserversorgung zu allen Biberburgen bauen?

Peter Bohner - peter.bohner@student.kit.edu

Lösung

- Modellierung als gewichteten, ungerichteten, zusammenhängenden Graphen:
  - Knoten: Burgen $B_1, \ldots, B_n$, sowie einen Knoten B_0 für Brunnen
  - Kanten: Streifen $S_1 \ldots S_m$ mit entsprechenden Gewichten $k_1, \ldots, k_m$
    Außerdem: Kanten von B_0 zu jedem anderen Knoten; Kante $\{B_0, B_i\}$ hat Gewicht t_i
  - Finden eines MST des Graphen (Prim oder Kruskal)
- Für jede Kante im MST zwischen zwei Burgen: baue Kanal zwischen diesen
- Für jede Kante $\{B_i, B_0\}$ im MST: baue Brunnen bei Burg B_i

Peter Bohner - peter.bohner@student.kit.edu

Graphalgorithmen - Wer kennt noch alle?

Karlsruher Institut für Technologie

Dijkstra

DFS

Toposort

Bellman-Ford

Floyd-Warshall

BFS

Prim

Kruskal

Bonusaufgabe: Wer kennt zu den Algorithmen die Laufzeit und was können diese?

Peter Bohner - peter.bohner@student.kit.edu

Graphalgorithmen

Algo	Laufzeit	Merkmale
BFS	$\mathcal{O}$(m)	SSSP auf ungewichteten Graphen und Zusammenhang
DFS	$\mathcal{O}$(m)	Brücken, Toposort und Zusammenhang
Dijkstra	$\Theta(m + n \log n)$**	SSSP auf positiv gewichteten Graphen
Bellman-Ford	$\mathcal{O}$(nm)	SSSP mit negativen Gewichten
Floyd-Warshall	$\Theta\left(n^3\right)$	APSP auf beliebigen Graphen
Prim	$\Theta(m + n \log n)$**	Minimale Spannbäume
Kruskal	$\Theta(m \log m + m \log^*(n))$	das Gleiche was Prim macht

** Mit Fibonacci Heap

* ist keine Fußnote sondern der iterierte Logarithmus

$n = |V|$
$m = |E|$

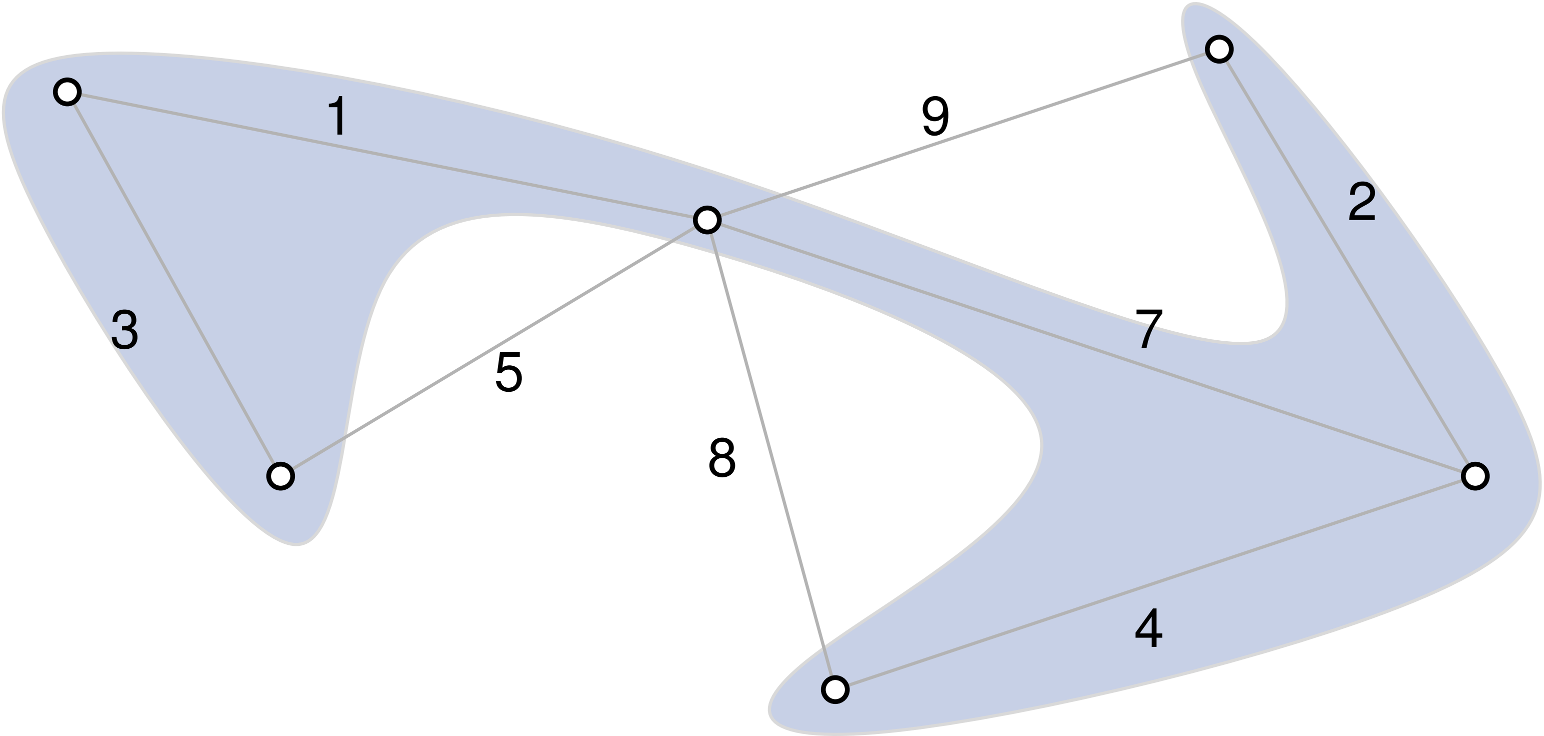
SSSP = **S**ingle **S**ource **S**hortest **P**ath
APSP = **A**ll Pair **S**hortest **P**ath

(eignet sich Prima für ein Cheat-Sheet :D)

Peter Bohner - peter.bohner@student.kit.edu

Union-Find - Motivation

Wir wollen Zusammenhangskomponenten verwalten (Bspw. für Kruskal)

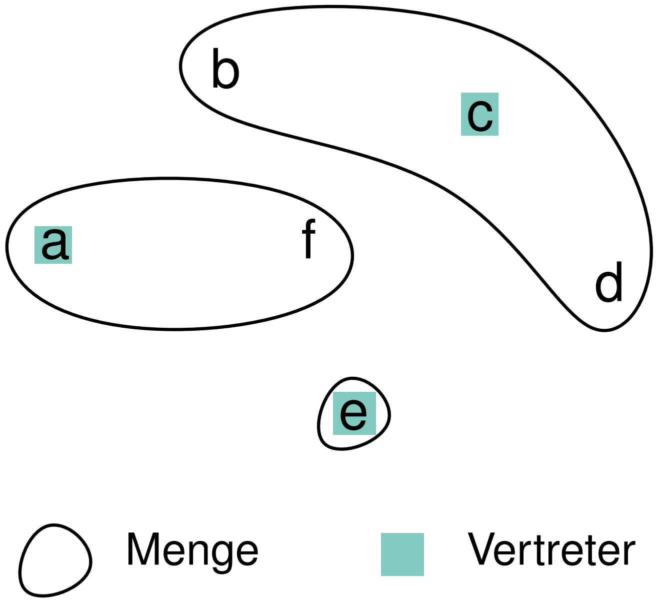


Union-Find

Verwalte Menge an disjunkten Mengen, anfangs einelementig

- **union**(x, y): Vereinige die Mengen, die x und y enthalten
- **find**(x): Liefere einen eindeutigen Vertreter der Menge, die x enthält
- **find**(x) == **find**(y): Prüft ob x und y in der gleichen Menge sind

Beispiel



union(a, f)

union(c, d)

find$(a) \rightarrow a$

find$(f) \rightarrow a$

find$(c) \rightarrow c$

union(b, d)

find$(b) \rightarrow c$

Peter Bohner - peter.bohner@student.kit.edu

Union by Rank

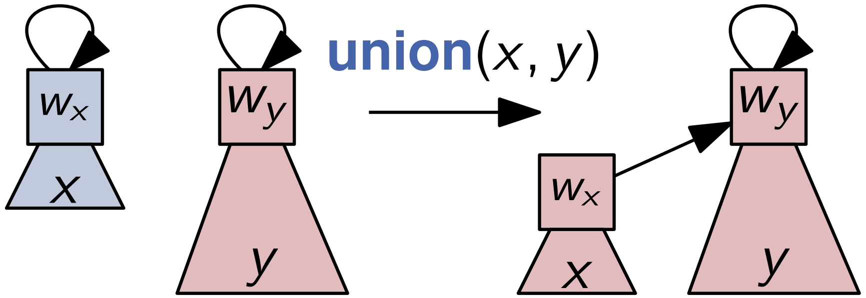
Jede Menge als “umgedrehten“ Baum speichern:

- Jedes Element zeigt auf Vorgänger auf Pfad zum Vertreter

- **find**(x): Gehe Pfad zum Vertreter
- **union**(x, y): **find** Vertreter von x und y

$\Rightarrow$ lasse weniger hohen auf den höheren zeigen

Höhe speichern und falls nötig aktualisieren!



Jeder dabei entstehende Baum mit Höhe h hat min. 2^h Elemente

Laufzeit von **find** ist also höchstens $\mathcal{O}(\log n)$

Peter Bohner - peter.bohner@student.kit.edu

Pfadkompression

Wir können bei **find**() ein kleines Bisschen extra Arbeit leisten, um die Laufzeit noch weiter zu beschleunigen

Immer wenn wir **find**() nutzen: Hänge jeden Knoten auf dem Pfad zur Wurzel direkt an die Wurzel

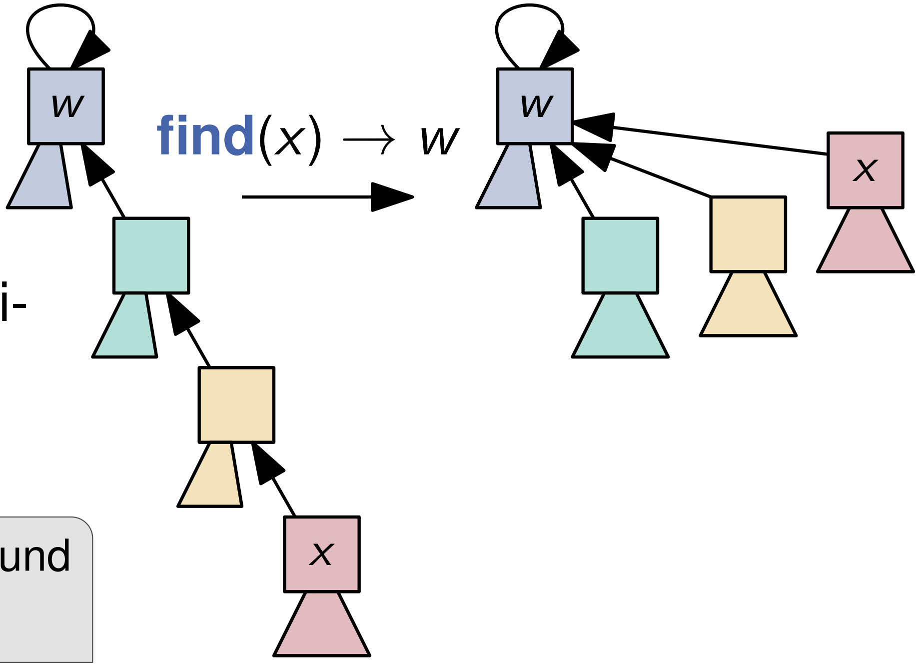
Laufzeit von **find**() bleibt gleich, denn wir wir haben nur konstanten zusätzlichen Aufwand pro Knoten

Aber alle zukünftigen **find**() Operationen werden schneller

Problem: Höhe schrumpft manchmal

Lösung: Statt der eigentlichen Höhe, speichere den Rang

- Erhöhe den Rang, wenn zwei Bäume gleichen Rangs vereinigt werden um eins



Eine beliebige Folge von **union** und **find** Operationen mit Union-by-Rank und Pfadkompression benötigt *amortisiert* $\mathcal{O}(\log^*(n))$ pro Operation.

Beweis: siehe VL

Peter Bohner - peter.bohner@student.kit.edu

Aufgaben

Karlsruher Institut für Technologie

Wie kann Union-Find verändert werden, sodass einzelne Elemente aus einer Menge in eine andere verschoben werden können, ohne den Rest zu beeinflussen?

- Blätter können immer umgehängt werden, ohne die restlichen Elemente zu beeinflussen $\rightarrow$ sorge dafür, dass alle Elemente Blätter sind $\rightarrow$ Füge extra Repräsentanten für jede (einelementige) Menge hinzu
- Zum Verschieben muss nur das entsprechende Blatt umgehängt werden

Verändere Union-Find, sodass auf die enthaltenen Elemente der Menge eines Elements in der selben Zeit wie der Repräsentant zugegriffen werden kann.

- Speichere bei jedem Repräsentanten alle Elemente in einer Liste
- Hänge bei **union** die Elemente des angehängten Knotens an die des (neuen) Repräsentanten an
- Alte Liste wird nicht mehr gebraucht und **concatenate** geht auf Listen in konstanter Zeit

Datenstrukturen aller Art

Arrays

Union-Find

Pointer

Listen

Dynamische Arrays

(2, 3)-Bäume

Priority-Queues/Heaps

Hashtabellen

Ihr solltet die Stärken und Schwächen einer Datenstruktur kennen!

Peter Bohner - peter.bohner@student.kit.edu

Datenstrukturen aller Art

Karlsruher Institut für Technologie

Pointer/Variablen

- Schreiben und Lesen in $\mathcal{O}(1)$
- Pointer können in $\mathcal{O}(1)$ *umgehangen* werden

(Dynamische) Arrays

- Zugriff auf beliebigen Index in $\mathcal{O}(1)$
- **pushBack**, **popBack** in *amortisiert* $\mathcal{O}(1)$
- Einfügen am Anfang oder in der Mitte in $\mathcal{O}(n)$

Listen

- Head und Tail Pointer (explizit dazuschreiben)
- Einfügen/Entfernen an beliebiger Stelle in $\mathcal{O}(1)$
- **splice**, **concatenate** in $\mathcal{O}(1)$
- Zugriff auf $i$tes Element in $\mathcal{O}(n)$

überall relevant

Peter Bohner - peter.bohner@student.kit.edu

Datenstrukturen aller Art

Hashtabellen

- Einfügen, Entfernen und Suchen in *erwartet* $\mathcal{O}(1)$

Anwendung

- Teilmenge großer Menge speichern
- Indizierung nicht möglich
- Look-Up-Table

Priotity Queues

- Üblicherweise als Heap realisiert
- **popMin** in $\Theta(\log n)$
- Einfügen und **decPrio** in amort. $\mathcal{O}(1)$ möglich

Anwendung

- Man benötigt immer das kleinste/größte Element
- Änderbare Prioritäten

(2, 3)-Bäume

- Einfügen, Entfernen und Suchen in $\Theta(\log n)$
- Der Alleskönner

Anwendung

- Dauerhafte Sortierung
- Sowohl Einfügen/Entfernen als auch häufiges Suchen benötigt

Datenstrukturen aller Art

Union-Find

- Kann **union** und **find** in amort. $\mathcal{O}(\alpha(n))$
- Ja das wars eigentlich

Anwendung

- Mengenvereinigung
- Prüfen ob zwei Elemente zur gleichen Menge gehören
- (Union-Find wird meist nur in Code-Challenges benutzt)

Graphen

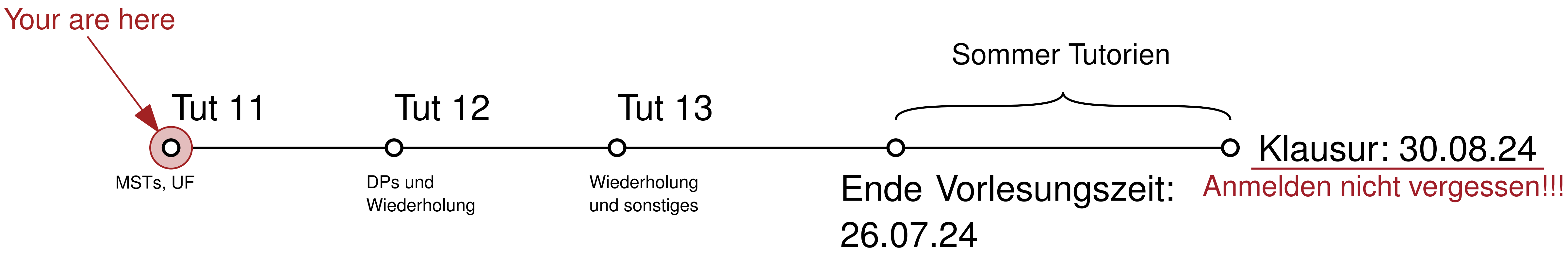
- Adjazenzen prüfen
- Implementierung über Adjazenzlisten oder Adjazenzmatrix

Technisch gesehen keine Datenstruktur, aber

- Viele Probleme lassen sich als Graph modellieren
- Wege/Routenplanung
- Relationen

Peter Bohner - peter.bohner@student.kit.edu

Roadmap



Falls ihr spezielle Themen nochmal wiederholen wollt, ruhig Bescheid geben (per Discord, E-Mail, auf einem Übungsblatt...)

Zusammenfassung

Was haben wir gemacht?

- Spannbäume
- Union Find

Worauf könnt ihr euch nächste Woche freuen?

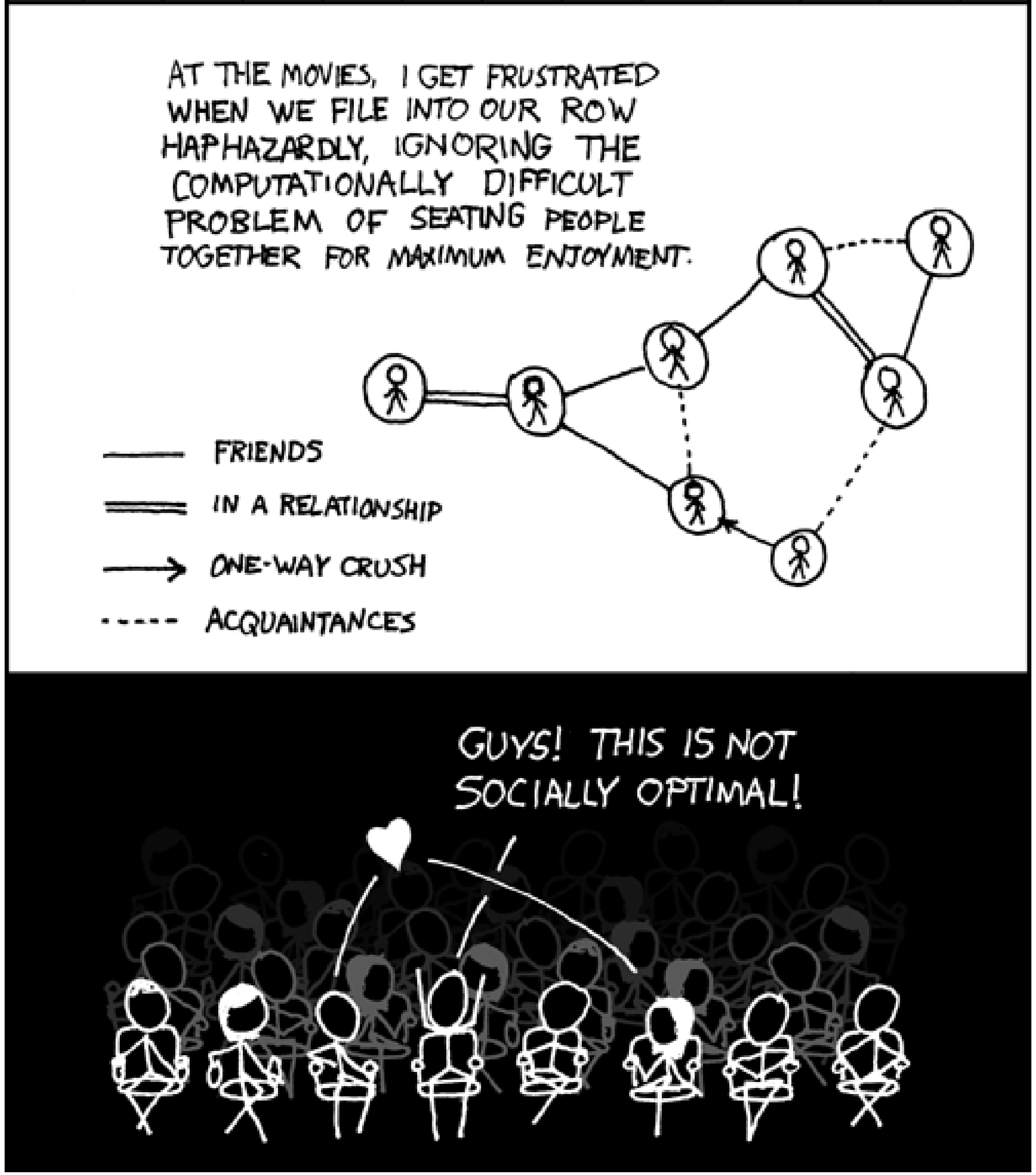
- Dynamische Programmierung

Peter Bohner - peter.bohner@student.kit.edu

Fragen?

Fragen!

Ende



https://xkcd.com/173/

Peter Bohner - peter.bohner@student.kit.edu