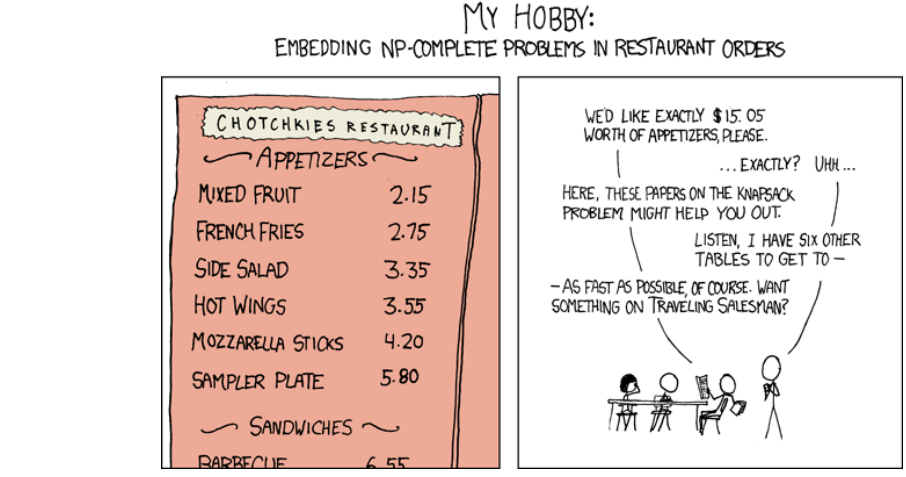
ETH zürich

D&A - Übungsstunde 12

Diese Folien basieren auf denjenigen der Vorlesung, wurden aber durch den Assistenten Adel Gavranović adaptiert und erweitert

Comic der Woche



xkcd

Übersicht

Heutiges Programm

Intro
Follow-up
Feedback zu [code]expert
Minimale Spannbäume
    Jarnik, Prim, und Dijkstra
    Kruskal
Live [code]expert
Max-Flow
Max-Flow von Hand
Alte Prüfungsaufgaben (Max-Flow)
Rundreiseproblem
Outro

n.ethz.ch/~agavranovic

- Link zum Material für die Übungsstunden
- Webseite des Assistenten
- Mail an Assistenten

1. Intro

Intro

Intro

- Endspurt!

2. Follow-up

Follow-up aus vorherigen Übungsstunden

Follow-up aus vorherigen Übungsstunden

- Die beiden alten Prüfungsaufgaben waren tatsächlich nur versehentlich noch in den Slides

Follow-up aus vorherigen Übungsstunden

- Die beiden alten Prüfungsaufgaben waren tatsächlich nur versehentlich noch in den Slides
- Verwirrung zur Breitensuche
  - Man geht bei der Breitensuche immer mit einer Ordnung (z.B. lexikografisch) vor

Follow-up aus vorherigen Übungsstunden

- Die beiden alten Prüfungsaufgaben waren tatsächlich nur versehentlich noch in den Slides
- Verwirrung zur Breitensuche
  - Man geht bei der Breitensuche immer mit einer Ordnung (z.B. lexikografisch) vor
  - D.h. erst As Nachfolger und dann Bs

Follow-up aus vorherigen Übungsstunden

- Die beiden alten Prüfungsaufgaben waren tatsächlich nur versehentlich noch in den Slides
- Verwirrung zur Breitensuche
  - Man geht bei der Breitensuche immer mit einer Ordnung (z.B. lexikografisch) vor
  - D.h. erst As Nachfolger und dann Bs
  - Um den Überblick zu behalten, lohnt es sich, einen Baum zu skizzieren

3. Feedback zu [code]expert

Allgemeines zu [code]expert

Allgemeines zu [code]expert

- Haben alle ihr Feedback?

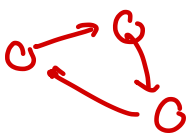
Allgemeines zu [code]expert

- Haben alle ihr Feedback?
- Haben alle ihre XP?

Task "Graph Traversals and Topological Sorting"

Task "Graph Traversals and Topological Sorting"

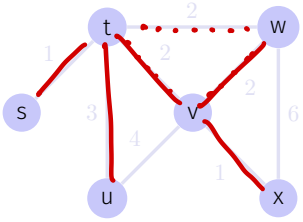
- Es heisst *Cycles* und nicht *Circles* oder *Loops*



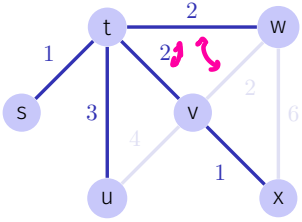
Fragen zu [code]expert eurerseits?

4. Minimale Spannbäume (MST)

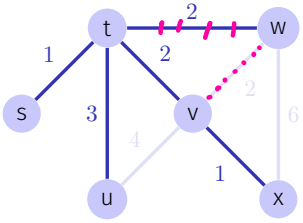
Minimale Spannbäume

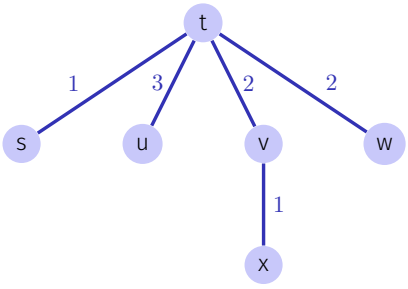


Minimale Spannbäume



Minimale Spannbäume





(Lösung ist nicht eindeutig.)

Jarnik-Prim-Dijkstra Algorithmus

- Findet minimalen Spannbaum.
- Startet von einem einzelnen Knoten und wächst.
- Verwendet einen Prioritätswarteschlange.
- Bewertet Kanten, nicht Pfade.

Algorithmus Jarnik-Prim-Dijkstra(G)

Input: Ein zusammenhängender, ungerichteter Graph $G = (V, E)$ mit Gewichten w
Output: Ein minimaler Spannbaum

Initialisiere $T = \emptyset$
Wähle arbiträr Knoten v_0 aus V
while $V \neq \emptyset$ **do**
    Wähle Kante (u, v) mit kleinstem Gewicht, so dass u in T und v in $V \setminus T$ ist
    Füge v zu T hinzu
    Entferne v aus V
return T

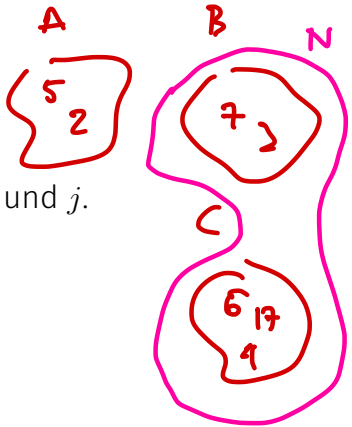
Unterschiede zum Dijkstra Algorithmus

- Jarnik-Prim-Dijkstra beurteilt Kanten, Dijkstra beurteilt Pfade.
- Jarnik-Prim-Dijkstra erstellt einen minimalen Spannbaum, Dijkstra findet den kürzesten Pfad.
- Jarnik-Prim-Dijkstra kann mit negativen Gewichten nicht umgehen, Dijkstra kann es unter bestimmten Bedingungen.

Union-Find Datenstruktur

In Union-Find Datenstruktur[1]

- Make-Set(i): Hinzufügen einer neuen Menge i.
- Find(e): Name i der Menge, welche e enthält.
- Union(i, j): Vereinigung der Mengen mit Namen i und j.



Find (5) → returns A

Union (B, C)

[1] alias: *disjoint-set data structure* und *merge–find set*

[2] Minium Spanning Tree

Union-Find Datenstruktur

In Union-Find Datenstruktur[1]

- Make-Set(i): Hinzufügen einer neuen Menge i.
- Find(e): Name i der Menge, welche e enthält.
- Union(i, j): Vereinigung der Mengen mit Namen i und j.

In MST[2]-Kruskal:

- Make-Set(i): Neuer Baum mit Wurzel i .
- Find(e): Finde Wurzel von e
- Union(i, j): Vereinigung der Bäume i und j.

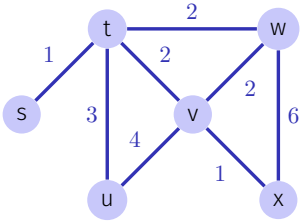
[1] alias: *disjoint-set data structure* und *merge–find set*
[2] Minium Spanning Tree

Algorithmus MST-Kruskal(G)

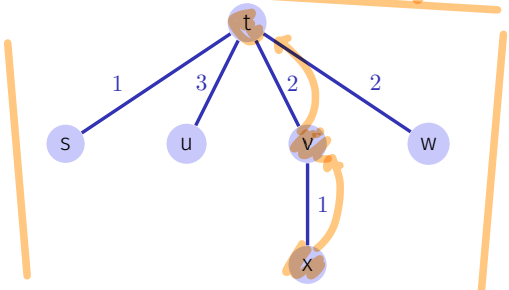
Input: Gewichteter Graph $G = (V, E, c)$
Output: Minimaler Spannbaum mit Kanten A.

Sortiere Kanten nach Gewicht $c(e_1) \leq ... \leq c(e_m)$
$A \leftarrow \emptyset$
for $k = 1$ **to** $|V|$ **do**
  MakeSet(k)
for $k = 1$ **to** m **do**
  $(u, v) \leftarrow e_k$
  if Find(u) $\neq$ Find(v) **then**
    Union(Find(u), Find(v))
    $A \leftarrow A \cup e_k$
return (V, A, c)

Repräsentation als Array



Index s t w v u x



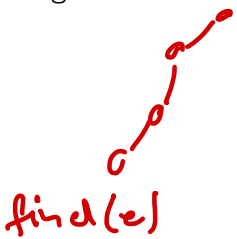
Index	s	t	u	v	w	x
Elter	t	t	t	t	t	v

Andere Verbesserung

Bei jedem Find alle Knoten direkt an den Wurzelknoten hängen.

Find(i):

$j \leftarrow i$
while $(p[i] \neq i)$ **do** $i \leftarrow p[i]$
while $(j \neq i)$ **do**
    $t \leftarrow j$
    $j \leftarrow p[j]$
    $p[t] \leftarrow i$
return i



Laufzeit: amortisiert *fast* konstant (Inverse der Ackermannfunktion).[3]

[3] Wird hier nicht vertieft.

Laufzeit des Kruskal Algorithmus

- Sortieren der Kanten: $\Theta(|E| \log |E|) = \Theta(|E| \log |V|)$. [4]
- Initialisieren der Union-Find Datenstruktur $\Theta(|V|)$
- $|E|\times$ Union(Find(x),Find(y)): $\mathcal{O}(|E| \log |E|) = \mathcal{O}(|E| \log |V|)$.

Insgesamt $\Theta(|E| \log |V|)$**.**

$|E|$: # Kanten

$|V|$: # Knoten

[4] da G zusammenhängend: $|V| \leq |E| \leq |V|^2$

Überblick

Problem: Minimale Spannbäume finden

- Mögliche Algorithmen dafür
  - Algorithmus von Jarnik, Prim, und Dijkstra
  - Algorithmus von Kruskal
    - benötigt die Union-Find Datenstruktur

(v. optimiert möglich)

Fragen/Unklarheiten?

5. Live [code]expert

Union-Find Experiments (auf [code]expert)

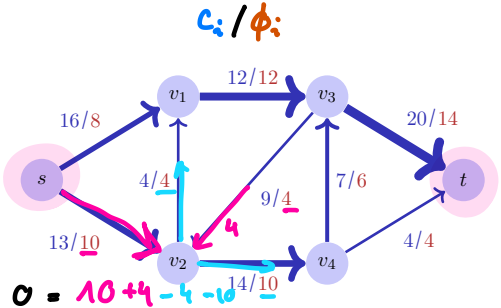
6. Max-Flow

Fluss

Ein **Fluss** $f : V \times V \to \mathbb{R}$ erfüllt folgende Bedingungen:

- **Kapazitätsbeschränkung**:
  Für alle $u, v \in V$: $f(u,v) \leq c(u,v)$.
- **Schiefsymmetrie**:
  Für alle $u, v \in V$: $f(u,v) = -f(v,u)$
- **Flusserhaltung**:
  Für alle $u \in V \setminus \{s,t\}$:

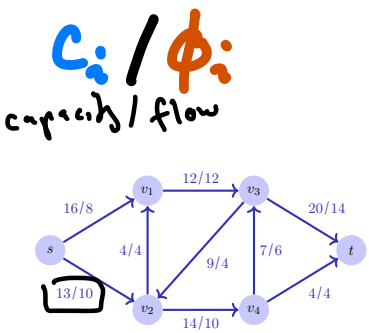
$$\sum_{v \in V} f(u,v) = 0.$$

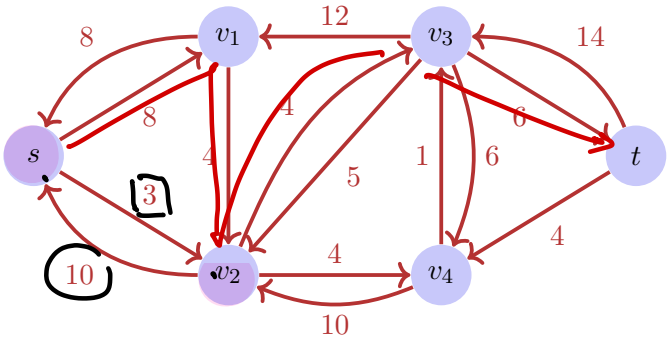


Wert w des Flusses:
$|f| = \sum_{v \in V} f(s,v)$.
Hier $|f| = 18$.

Restnetzwerk

Restnetzwerk G_f gegeben durch alle Kanten mit Restkapazität:



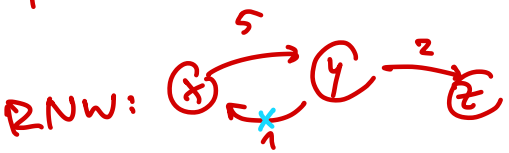


Restnetzwerke haben dieselben Eigenschaften wie Flussnetzwerke, ausser dass antiparallele Kanten zugelassen sind.

Erweiterungspfade

Erweiterungspfad p: einfacher Pfad von s nach t im Restnetzwerk G_f.

Restkapazität $c_f(p) = \min\{c_f(u,v) : (u,v) \text{ Kante in } p\}$

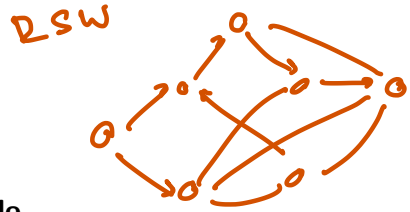


Algorithmus Ford-Fulkerson(G, s, t)

Input: Flussnetzwerk $G = (V, E, c)$
Output: Maximaler Fluss f.

for $(u, v) \in E$ **do**
  $f(u, v) \leftarrow 0$

while Existiert Pfad $p : s \rightsquigarrow t$ im Restnetzwerk G_f **do**
  $c_f(p) \leftarrow \min\{c_f(u, v) : (u, v) \in p\}$
  foreach $(u, v) \in p$ **do**
    if $(u, v) \in E$ **then**
      $f(u, v) \leftarrow f(u, v) + c_f(p)$
    else
      $f(v, u) \leftarrow f(u, v) - c_f(p)$



Edmonds-Karp Algorithmus

Wähle in der Ford-Fulkerson-Methode zum Finden eines Pfades in G_f jeweils einen Erweiterungspfad kürzester Länge (z.B. durch Breitensuche).

Theorem 1

Wenn der Edmonds-Karp Algorithmus auf ein ganzzahliges Flussnetzwerk $G = (V, E)$ mit Quelle s und Senke t angewendet wird, dann ist die Gesamtanzahl der durch den Algorithmus angewendete Flusserhöhungen in $\mathcal{O}(|V| \cdot |E|)$

$\Rightarrow$ ***Gesamte asymptotische Laufzeit:*** $\mathcal{O}(|V| \cdot |E|^2)$

Max-Flow Min-Cut Theorem

Theorem 2

Wenn f ein Fluss in einem Flussnetzwerk $G = (V, E, c)$ mit Quelle s und Senke t is, dann sind folgende Aussagen äquivalent:

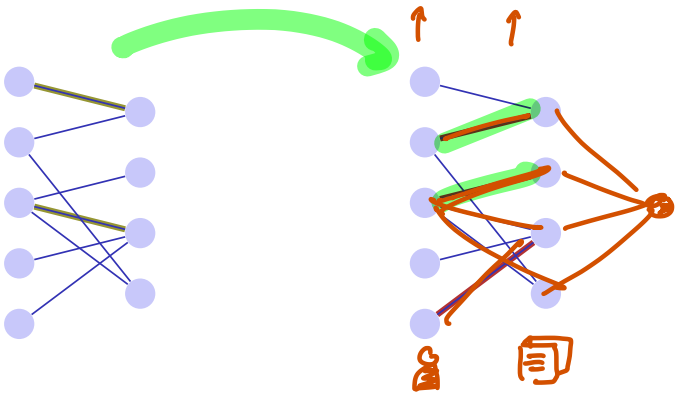
1. *f ist ein maximaler Fluss in G*
2. *Das Restnetzwerk G_f enthält keine Erweiterungspfade*
3. *Es gilt $|f| = c(S, T)$ für einen Schnitt (S, T) von G.*

Anwendung: Maximales bipartites Matching

Gegeben: bipartiter ungerichteter Graph $G = (V, E)$.

Matching M: $M \subseteq E$ so dass $|\{m \in M : v \in m\}| \leq 1$ für alle $v \in V$.

Maximales Matching M: Matching M, so dass $|M| \geq |M'|$ für jedes Matching M'.

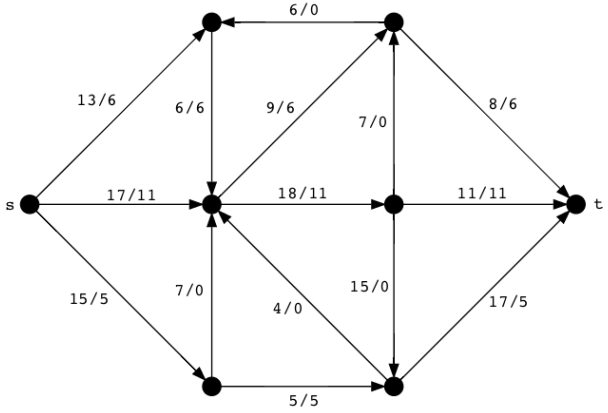


Fragen/Unklarheiten?

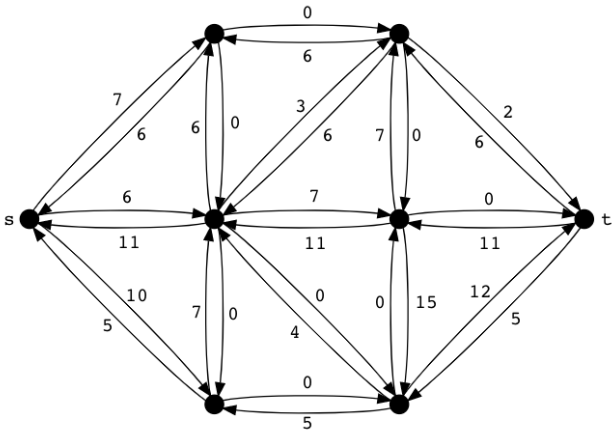
7. Max-Flow von Hand

Maximaler Fluss von Hand

Dieser Graph zeigt ein Flussdiagramm, das keinen maximalen Fluss aufweist. Führe eine Iteration des Ford-Fulkerson-Algorithmus aus, um den maximalen Fluss zu finden.



Maximaler Fluss von Hand (Lösung)



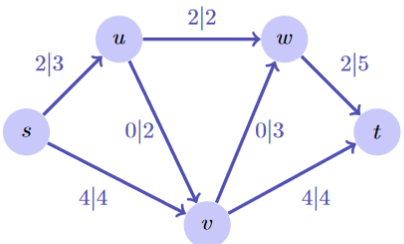
Update nicht gezeichnet, da nicht eindeutig!

8. Alte Prüfungsaufgaben (Max-Flow)

Beispiel Prüfungsaufgabe

Gegeben ist das folgende Flussnetzwerk G mit Quelle s und Senke t. Die einzelnen Kapazitäten c_i und Flüsse ϕ_i sind an den Kanten angegeben als $\phi_i|c_i$. Geben Sie den Wert des Flusses f an.

Provided in the following is a flow network G with source s and sink t. Capacities c_i and flows ϕ_i are provided at the edges as $\phi_i|c_i$. Provide the value of the flow f.

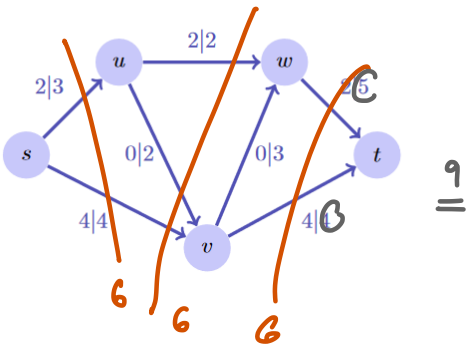


$|f| =$

Beispiel Prüfungsaufgabe

Gegeben ist das folgende Flussnetzwerk G mit Quelle s und Senke t. Die einzelnen Kapazitäten c_i und Flüsse ϕ_i sind an den Kanten angegeben als $\phi_i|c_i$. Geben Sie den Wert des Flusses f an.

Provided in the following is a flow network G with source s and sink t. Capacities c_i and flows ϕ_i are provided at the edges as $\phi_i|c_i$. Provide the value of the flow f.

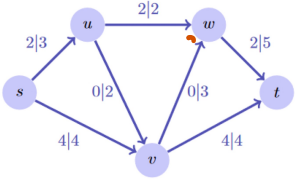


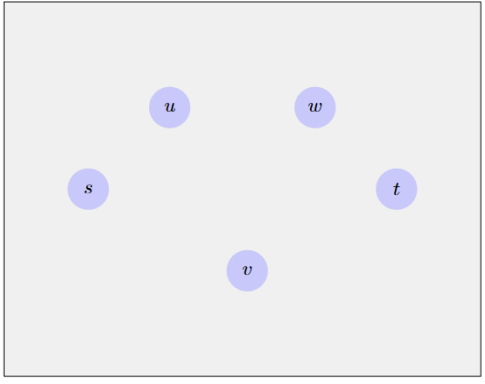
$|f| =$ 6

Beispiel Prüfungsaufgabe

Zeichnen Sie nun das Restnetzwerk G_f zu obigem Fluss und markieren Sie darin einen Erweiterungspfad p. Geben Sie den Wert $c_f(p)$ der Restkapazität des Erweiterungspfades p im Restnetzwerk G_f an.

Draw the residual network G_f to the flow above and mark an augmenting path p. Provide the rest capacity $c_f(p)$ of the path p in the rest network G_f.



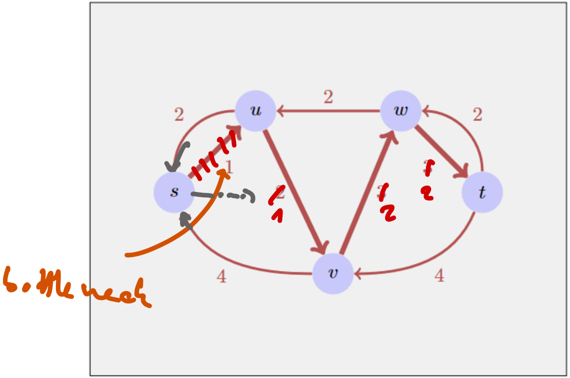


$|c_f(p)| =$

Beispiel Prüfungsaufgabe

Zeichnen Sie nun das Restnetzwerk G_f zu obigem Fluss und markieren Sie darin einen Erweiterungspfad p. Geben Sie den Wert $c_f(p)$ der Restkapazität des Erweiterungspfades p im Restnetzwerk G_f an.

Draw the residual network G_f to the flow above and mark an augmenting path p. Provide the rest capacity $c_f(p)$ of the path p in the rest network G_f.



$|c_f(p)| = $ 1

Beispiel Prüfungsaufgabe

Woran erkennen Sie, ob Sie den maximalen Fluss gefunden haben?

How do you see if you have found the maximum flow?

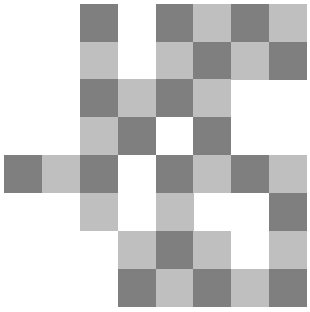
Beispiel Prüfungsaufgabe

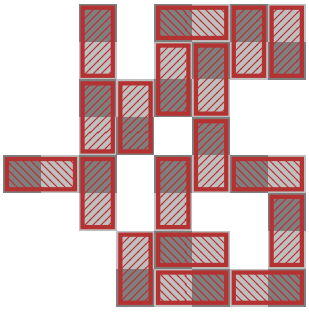
Woran erkennen Sie, ob Sie den maximalen Fluss gefunden haben?

How do you see if you have found the maximum flow?

Found the maximum flow if:
The residual network does not have any more augmenting path.
Alternative: Identify a cut with $|c(S,T)| = |f|$.

Max Flow Question





Gegeben sei ein n × n-Schachbrett ohne einige Felder. Beschreiben Sie einen effizienten Algorithmus, um herauszufinden, ob das Spielbrett vollständig mit Dominosteinen bedeckt werden kann. Modellieren Sie das Problem dann als Flussproblem.

9. Rundreiseproblem

Rundreiseproblem

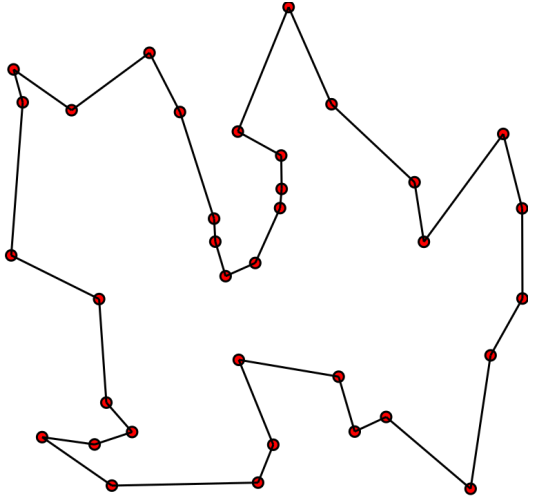
Problem

Gegeben ist eine Karte und eine Liste von Städten. Welches ist die kürzeste Route, die jede Stadt einmal besucht und in die ursprüngliche Stadt zurückkehrt?

Mathematical model

Auf einem ungerichteten, gewichteten Graph G ist nach dem Kreis gesucht, welcher jede der Knoten von G genau einmal enthält und die kleinste Kantensumme aufweist.

Rundreiseproblem



Rundreiseproblem

- Es ist kein Polynomialzeitalgorithmus zum Lösen des Problems bekannt.
- Es gibt verschiedene heuristische Algorithmen. Diese liefern oft nicht die optimale Lösung.

Rundreiseproblem

- Der heuristische Algorithmus, den Sie auf CodeExpert implementieren sollen (*The Travelling Student*) benutzt einen Minimalen Spannbaum:
  1. Berechne den Minimalen Spannbaum M
  2. Mache eine Tiefensuche auf M
- Der Algorithmus ist eine 2-Approximation. Das bedeutet, dass er eine Lösung liefert, die maximal 2 mal die Kosten einer optimalen Lösung aufweist.
- Der Algorithmus geht von einem vollständigen Graphen $G = (V, E, c)$ aus, auf dem die Dreiecksungleichung gilt:
  $c(v, w) \leq c(v, x) + c(x, w) \; \forall \; v, w, x \in V$

10. Outro

Allgemeine Fragen?

D&A-PVK im Sommer

Informationen

Datum	20.06.23 - 23.06.23
Uhrzeit	08:00 - 12:00
Raum	NO E 11
Kosten	CHF 40.-

Anmeldung nötig

▸ Anmeldung

Bis zum nächsten Mal

Schönes Wochenende!