# Exercise Session 10 – Graphs & Paths

**Data Structures and Algorithms**

*These slides are based on those of the lecture, but were adapted and extended by the teaching assistant Adel Gavranović*

# Today's Schedule

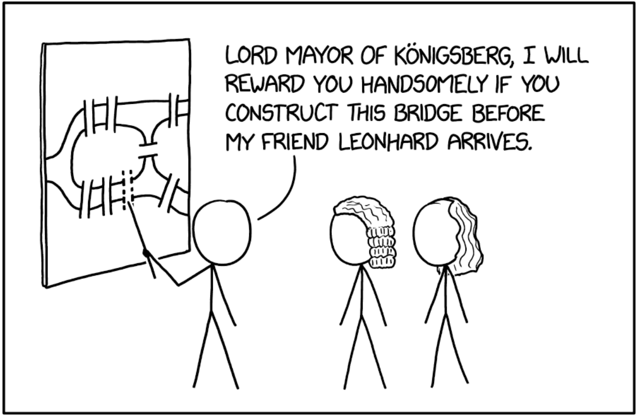
Intro
Follow-up
Feedback for **code** expert
Learning Objectives
Minor Recap
Theory Recap
- Shortest Paths
- All Pairs Shortest Paths
- Minimum Spanning Trees

Code-Expert Exercise
TSP
Old Exam Question
Outro

`n.ethz.ch/~agavranovic`

- Exercise Session Material
- Adel's Webpage
- Mail to Adel

# Comic of the Week




I TRIED TO USE A TIME MACHINE TO CHEAT ON MY ALGORITHMS FINAL BY PREVENTING GRAPH THEORY FROM BEING INVENTED.

xkcd

# 1. Intro

# Intro

- My throat is still a little sore
- New date for the session next week
  - Thu next week is a holiday
  - Alternatives should be listed on the course page soon
  - My session:
    - Wed 8th May
    - 16 - 18
    - CHN G 42

# 2. Follow-up

# Follow-up from last exercise session

- **Quiz: Runtimes of Simple Operations (s. 13)**
  - There are differences between undirected and directed graph's runtimes[1] especially for the "find all neighbours" operation
- **Quiz #3 (s. 17)**
  - Was arguably a little too complicated for a tiny quiz question
- **Dijkstra Turning Exponential (s. 46)**
  - Basically, every possible bit patters is a new possible combination that the algorithm has to check, i.e. exponentially many paths, thus $\mathcal{O}(2^n)$

[1] think about it, but don't worry too much about it

# 3. Feedback for **code** expert

# code expert

**Master Solution for "Trees"**

- Is on my homepage now
- If you notice errors, let me know

**Non-deterministic Grading for "Amazing Mazes I"**

- Don't be alarmed if you get a different grading for the same code
- (yes, just submit the better graded one)
- Long story short: the mazes are generated pseudo-randomly and that might cause *some* test to take longer than intended, yielding a virtual timer error
- Even the Master Solution suffered from this

# Questions regarding **code** expert from your side?

# 4. Learning Objectives

# Learning Objectives

Understand how and why…

- ☐ …the A* algorithm
- ☐ …the Bellman-Ford algorithm
- ☐ …the Floyd-Warshall algorithm
- ☐ …the Jarnik-Prim-Dijkstra algorithm
- ☐ …Kruskal's algorithm

works and when to use it

# 5. Minor Recap

# Minor Recap

**Quick recap on all of these**

- Heuristic
- Transitive Closure
- Bellman-Ford Algorithm
- Floyd-Warshall Algorithm

# 6. Theory Recap

# 6.1 Shortest Paths

# A*-Algorithm($G, s, t, \hat{h}$)

**Input:** Positively weighted Graph $G = (V, E, c)$, starting point $s \in V$, end point $t \in V$, estimate $\widehat{h}(v) \leq \delta(v, t)$
**Output:** Existence and value of a shortest path from $s$ to $t$

**foreach** $u \in V$ **do**
  $d[u] \leftarrow \infty$; $\widehat{f}[u] \leftarrow \infty$; $\pi[u] \leftarrow$ null

$d[s] \leftarrow 0$; $\widehat{f}[s] \leftarrow \widehat{h}(s)$; $R \leftarrow \{s\}$; $M \leftarrow \{\}$
**while** $R \neq \emptyset$ **do**
  $u \leftarrow$ ExtractMin$_{\widehat{f}}(R)$; $M \leftarrow M \cup \{u\}$
  **if** $u = t$ **then return** success
  **foreach** $v \in N^+(u)$ with $d[v] > d[u] + c(u, v)$ **do**
    $d[v] \leftarrow d[u] + c(u, v)$; $\widehat{f}[v] \leftarrow d[v] + \widehat{h}(v)$; $\pi[v] \leftarrow u$
    $R \leftarrow R \cup \{v\}$; $M \leftarrow M - \{v\}$

**return** failure

# Properties

- The A*-Algorithm is an extension of the Dijkstra algortihm by a distance heuristic $\hat{h}$.
- A* is Dijkstra if $\hat{h} \equiv 0$
- underestimation: $\forall v \in V: \hat{h}(v) \leq \delta(v,t)$
  If $\hat{h}$ underestimates the real distance, the algorithm works correctly.
- Monotonicity: $\forall (u,u') \in E: \widehat{h}(u') \leq \widehat{h}(u) + c(u',u)$
  If $\hat{h}$ is monotone in addition, then the algorithm works efficiently.

# General Weighted Graphs

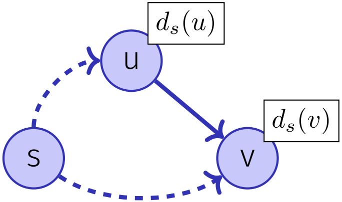
Relax$(u, v)$ $(u, v \in V$, $(u, v) \in E)$

**if** $d_s(v) > d_s(u) + c(u, v)$ **then**

  $d_s(v) \leftarrow d_s(u) + c(u, v)$

  **return** true

**return** false




Problem: cycles with negative weights can shorten the path, a shortest path is not guaranteed to exist.

# Dynamic Programming Approach (Bellman)

Induction over number of edges $d_s[i, v]$:
Shortest path from $s$ to $v$ via maximally $i$ edges.

$$d_s[i, v] = \min\{d_s[i-1, v], \min_{(u,v)\in E}(d_s[i-1, u] + c(u, v))$$

$$d_s[0, s] = 0, d_s[0, v] = \infty \ \forall v \neq s.$$

# Algorithm Bellman-Ford($G, s$)

**Input:** Graph $G = (V, E, c)$, starting point $s \in V$
**Output:** If return value true, minimal weights $d$ for all shortest paths from $s$, otherwise no shortest path.

**foreach** $u \in V$ **do**
    $d_s[u] \leftarrow \infty$; $\pi_s[u] \leftarrow$ null
$d_s[s] \leftarrow 0$;
**for** $i \leftarrow 1$ **to** $|V|$ **do**
    $f \leftarrow$ false
    **foreach** $(u, v) \in E$ **do**
        $f \leftarrow f \vee \mathrm{Relax}(u, v)$
    **if** $f =$ false **then return** true
**return** false;

# 6.2 All Pairs Shortest Paths

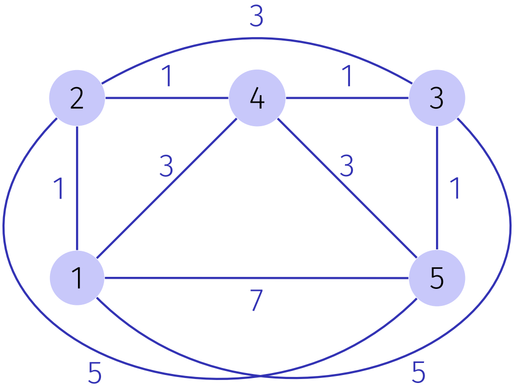
# DP Algorithm Floyd-Warshall($G$)

**Input:** Acyclic Graph $G = (V, E, c)$
**Output:** Minimal weights of all paths $d$
$d^0 \leftarrow c$
**for** $k \leftarrow 1$ **to** $|V|$ **do**
  **for** $i \leftarrow 1$ **to** $|V|$ **do**
    **for** $j \leftarrow 1$ **to** $|V|$ **do**
      $d^k(v_i, v_j) = \min\{d^{k-1}(v_i, v_j), d^{k-1}(v_i, v_k) + d^{k-1}(v_k, v_j)\}$

Runtime: $\Theta(|V|^3)$

Remark: Algorithm can be executed with a single matrix $d$ (in place).

# Example




adjacency matrix $M = c$

$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ 1 & 0 & 1 & 5 & 3 & 7 \\ 2 & 1 & 0 & 3 & 1 & 5 \\ 3 & 5 & 3 & 0 & 1 & 1 \\ 4 & 3 & 1 & 1 & 0 & 3 \\ 5 & 7 & 5 & 1 & 3 & 0 \end{array}$$

# Example

$$
\begin{array}{ccccc}
k=1 & k=2 & k=3 & k=4 & k=5 \\
\begin{matrix} 0&1&5&3&7\\ 1&0&3&1&5\\ 5&3&0&1&1\\ 3&1&1&0&3\\ 7&5&1&3&0 \end{matrix} &
\begin{matrix} 0&1&5&3&7\\ 1&0&3&1&5\\ 5&3&0&1&1\\ 3&1&1&0&3\\ 7&5&1&3&0 \end{matrix} &
\begin{matrix} 0&1&4&2&6\\ 1&0&3&1&5\\ 4&3&0&1&1\\ 2&1&1&0&3\\ 6&5&1&3&0 \end{matrix} &
\begin{matrix} 0&1&4&2&5\\ 1&0&3&1&4\\ 4&3&0&1&1\\ 2&1&1&0&2\\ 5&4&1&2&0 \end{matrix} &
\begin{matrix} 0&1&3&2&4\\ 1&0&2&1&3\\ 3&2&0&1&1\\ 2&1&1&0&2\\ 4&3&1&2&0 \end{matrix} \\
d^0 & d^1 & d^2 & d^3 & d^4
\end{array}
$$

$$
\begin{array}{ccccc}
\begin{matrix} 0&1&5&3&7\\ 1&0&3&1&5\\ 5&3&0&1&1\\ 3&1&1&0&3\\ 7&5&1&3&0 \end{matrix} &
\begin{matrix} 0&1&\mathbf{4}&\mathbf{2}&\mathbf{6}\\ 1&0&3&1&5\\ \mathbf{4}&3&0&1&1\\ \mathbf{2}&1&1&0&3\\ \mathbf{6}&5&1&3&0 \end{matrix} &
\begin{matrix} 0&1&4&2&\mathbf{5}\\ 1&0&3&1&\mathbf{4}\\ 4&3&0&1&1\\ 2&1&1&0&\mathbf{2}\\ \mathbf{5}&\mathbf{4}&1&\mathbf{2}&0 \end{matrix} &
\begin{matrix} 0&1&\mathbf{3}&2&\mathbf{4}\\ 1&0&\mathbf{2}&1&\mathbf{3}\\ \mathbf{3}&\mathbf{2}&0&1&1\\ 2&1&1&0&2\\ \mathbf{4}&\mathbf{3}&1&2&0 \end{matrix} &
\begin{matrix} 0&1&3&2&4\\ 1&0&2&1&3\\ 3&2&0&1&1\\ 2&1&1&0&2\\ 4&3&1&2&0 \end{matrix} \\
d^1 & d^2 & d^3 & d^4 & d^5
\end{array}
$$

# Shortest Path for each Pair?

$$M \qquad\qquad D := d^5 \qquad\qquad\qquad\qquad D''$$

$$\begin{matrix} 0 & 1 & 5 & 3 & 7 \\ 1 & 0 & 3 & 1 & 5 \\ 5 & 3 & 0 & 1 & 1 \\ 3 & 1 & 1 & 0 & 3 \\ 7 & 5 & 1 & 3 & 0 \end{matrix} \qquad \begin{matrix} 0 & 1 & 3 & 2 & 4 \\ 1 & 0 & 2 & 1 & 3 \\ 3 & 2 & 0 & 1 & 1 \\ 2 & 1 & 1 & 0 & 2 \\ 4 & 3 & 1 & 2 & 0 \end{matrix} \qquad \begin{matrix} 0 & 1 & 3 & 2 & 4 \\ 1 & 0 & 2 & 1 & 3 \\ 3 & 2 & 0 & 1 & 1 \\ 2 & 1 & 1 & 0 & 2 \\ 4 & 3 & 1 & 2 & 0 \end{matrix} \qquad \begin{matrix} 0 & 1 & 3 & 2 & 4 \\ 1 & 0 & 2 & 1 & 3 \\ 3 & 2 & 0 & 1 & 1 \\ 2 & 1 & 1 & 0 & 2 \\ 4 & 3 & 1 & 2 & 0 \end{matrix}$$

**Question**: Can we use the computed matrix $D$ to determine the shortest path between each pair of nodes?

Direct connections $i \to j$ where $M[i,j] = D[i,j]$ (cf markings in $D'$ above)

Could try to run the algorithm backwards. Example $1 \to 3$ above in $D''$. Find, with decreasing $k$, the first fitting candidate.

Complicated and inefficient.

# Idea

Memorize the best $k$ in the algorithm for each node pair $(i, j)$.
Start with matrix of existing direct connections (edges)

# Example

$k = 1$

$$B = \begin{array}{ccccc} 0 & 1 & 5 & 3 & 7 \\ 1 & 0 & 3 & 1 & 5 \\ 5 & 3 & 0 & 1 & 1 \\ 3 & 1 & 1 & 0 & 3 \\ 7 & 5 & 1 & 3 & 0 \end{array} \qquad K = \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \end{array}$$

$k = 2$

$$\begin{array}{ccccc} 0 & 1 & \mathbf{4} & \mathbf{2} & \mathbf{6} \\ 1 & 0 & 3 & 1 & 5 \\ \mathbf{4} & 3 & 0 & 1 & 1 \\ \mathbf{2} & 1 & 1 & 0 & 3 \\ \mathbf{6} & 5 & 1 & 3 & 0 \end{array} \qquad \begin{array}{ccccc} 1 & 2 & \mathbf{2} & \mathbf{2} & \mathbf{2} \\ 1 & 2 & 3 & 4 & 5 \\ \mathbf{2} & 2 & 3 & 4 & 5 \\ \mathbf{2} & 2 & 3 & 4 & 5 \\ \mathbf{2} & 2 & 3 & 4 & 5 \end{array}$$

$k = 3$

$$\begin{array}{ccccc} 0 & 1 & 4 & 2 & \mathbf{5} \\ 1 & 0 & 3 & 1 & \mathbf{4} \\ 4 & 3 & 0 & 1 & 1 \\ 2 & 1 & 1 & 0 & \mathbf{2} \\ \mathbf{5} & \mathbf{4} & 1 & \mathbf{2} & 0 \end{array} \qquad \begin{array}{ccccc} 1 & 2 & 2 & 2 & \mathbf{3} \\ 1 & 2 & 3 & 4 & \mathbf{3} \\ 2 & 2 & 3 & 4 & 5 \\ 2 & 2 & 3 & 4 & \mathbf{3} \\ \mathbf{3} & \mathbf{3} & 3 & \mathbf{3} & 5 \end{array}$$

$k = 4$

$$\begin{array}{ccccc} 0 & 1 & \mathbf{3} & 2 & \mathbf{4} \\ 1 & 0 & \mathbf{2} & 1 & \mathbf{3} \\ \mathbf{3} & \mathbf{2} & 0 & 1 & 1 \\ 2 & 1 & 1 & 0 & 2 \\ \mathbf{4} & \mathbf{3} & 1 & 2 & 0 \end{array} \qquad \begin{array}{ccccc} 1 & 2 & \mathbf{4} & 2 & \mathbf{4} \\ 1 & 2 & \mathbf{4} & 4 & \mathbf{4} \\ \mathbf{4} & \mathbf{4} & 3 & 4 & 5 \\ 2 & 2 & 3 & 4 & 3 \\ \mathbf{4} & \mathbf{4} & 3 & 3 & 5 \end{array}$$

$k = 5$

$$\begin{array}{ccccc} 0 & 1 & 3 & 2 & 4 \\ 1 & 0 & 2 & 1 & 3 \\ 3 & 2 & 0 & 1 & 1 \\ 2 & 1 & 1 & 0 & 2 \\ 4 & 3 & 1 & 2 & 0 \end{array} \qquad \begin{array}{ccccc} 1 & 2 & 4 & 2 & 4 \\ 1 & 2 & 4 & 4 & 4 \\ 4 & 4 & 3 & 4 & 5 \\ 2 & 2 & 3 & 4 & 3 \\ 4 & 4 & 3 & 3 & 5 \end{array}$$

# Example

$$\begin{array}{c|ccccc} K & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 1 & 2 & 4 & 2 & 4 \\ 2 & 1 & 2 & 4 & 4 & 4 \\ 3 & 4 & 4 & 3 & 4 & 5 \\ 4 & 2 & 2 & 3 & 4 & 3 \\ 5 & 4 & 4 & 3 & 3 & 5 \end{array}$$

How to read this matrix $K$? Example path $1 \to 5$:

- Path $1 \to 5$ goes via node $4$.
- Path $1 \to 4$ goes via node $2$.
- Path $4 \to 5$ goes via node $3$.
- Paths $1 \to 2$ and $2 \to 4$ are direct.
- Paths $4 \to 3$ and $3 \to 5$ are direct.

Overall

$$1 \xrightarrow{4} 5 \quad \Rightarrow \quad 1 \xrightarrow{2} 4 \xrightarrow{3} 5 \quad \Rightarrow \quad 1 \to 2 \to 4 \to 3 \to 5$$

Reconstruction via Recursion. Alternative? Store descenden in the algorithm

# Example

$$
\begin{array}{cccccc}
 & k=1 & k=2 & k=3 & k=4 & k=5 \\
B &
\begin{array}{ccccc} 0&1&5&3&7\\ 1&0&3&1&5\\ 5&3&0&1&1\\ 3&1&1&0&3\\ 7&5&1&3&0 \end{array} &
\begin{array}{ccccc} 0&1&4&2&6\\ 1&0&3&1&5\\ 4&3&0&1&1\\ 2&1&1&0&3\\ 6&5&1&3&0 \end{array} &
\begin{array}{ccccc} 0&1&4&2&5\\ 1&0&3&1&4\\ 4&3&0&1&1\\ 2&1&1&0&2\\ 5&4&1&2&0 \end{array} &
\begin{array}{ccccc} 0&1&3&2&4\\ 1&0&2&1&3\\ 3&2&0&1&1\\ 2&1&1&0&2\\ 4&3&1&2&0 \end{array} &
\begin{array}{ccccc} 0&1&3&2&4\\ 1&0&2&1&3\\ 3&2&0&1&1\\ 2&1&1&0&2\\ 4&3&1&2&0 \end{array} \\
K &
\begin{array}{ccccc} 1&2&3&4&5\\ 1&2&3&4&5\\ 1&2&3&4&5\\ 1&2&3&4&5\\ 1&2&3&4&5 \end{array} &
\begin{array}{ccccc} 1&2&2&2&2\\ 1&2&3&4&5\\ 2&2&3&4&5\\ 2&2&3&4&5\\ 2&2&3&4&5 \end{array} &
\begin{array}{ccccc} 1&2&2&2&2\\ 1&2&3&4&3\\ 2&2&3&4&5\\ 2&2&3&4&3\\ 3&3&3&3&5 \end{array} &
\begin{array}{ccccc} 1&2&2&2&2\\ 1&2&4&4&4\\ 4&4&3&4&5\\ 2&2&3&4&3\\ 3&3&3&3&5 \end{array} &
\begin{array}{ccccc} 1&2&2&2&2\\ 1&2&4&4&4\\ 4&4&3&4&5\\ 2&2&3&4&3\\ 3&3&3&3&5 \end{array}
\end{array}
$$

# Comparison of the approaches

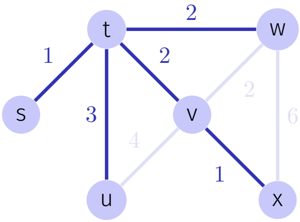
| Algorithm | | | Runtime |
|---|---|---|---|
| Dijkstra (Heap) | $c_v \geq 0$ | 1:n | $\mathcal{O}(\lvert E\rvert \log \lvert V\rvert)$ |
| Dijkstra (Fibonacci-Heap) | $c_v \geq 0$ | 1:n | $\mathcal{O}(\lvert E\rvert + \lvert V\rvert \log \lvert V\rvert)$ * |
| Bellman-Ford | | 1:n | $\mathcal{O}(\lvert E\rvert \cdot \lvert V\rvert)$ |
| Floyd-Warshall | | n:n | $\Theta(\lvert V\rvert^3)$ |
| Johnson | | n:n | $\mathcal{O}(\lvert V\rvert \cdot \lvert E\rvert \cdot \log \lvert V\rvert)$ |
| Johnson (Fibonacci-Heap) | | n:n | $\mathcal{O}(\lvert V\rvert^2 \log \lvert V\rvert + \lvert V\rvert \cdot \lvert E\rvert)$ * |

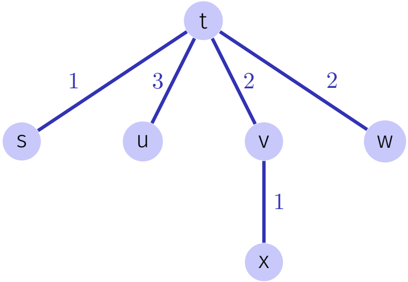
* amortized

Johnson (not explained this year) is better than Floyd-Warshall only for sparse graphs ($|E| \approx \Theta(|V|)$).

# 6.3 Minimum Spanning Trees

# Minimum Spanning Trees




(Solution is not unique.)

# Jarnik-Prim-Dijkstra Algorithm

- Finds a minimum spanning tree.
- Starts from a single node and grows.
- Uses a priority queue.
- Evaluates edges, not paths.

# Algorithm Jarnik-Prim-Dijkstra($G$)

**Input:** A connected, undirected graph $G = (V, E)$ with weights $w$
**Output:** A minimum spanning tree $T$

Initialize $T = \emptyset$
Choose arbitrary vertex $v_0$ from $V$
**while** $V \neq \emptyset$ **do**
    Choose edge $(u, v)$ with smallest weight such that $u$ is in $T$ and $v$ is in $V - T$
    Add $v$ to $T$
    Remove $v$ from $V$

**return** $T$

# Differences from Dijkstra's Algorithm

- Jarnik-Prim-Dijkstra evaluates edges. Dijkstra evaluates paths.
- Jarnik-Prim-Dijkstra creates a minimum spanning tree. Dijkstra finds the shortest path.
- Jarnik-Prim-Dijkstra cannot handle negative weights. Dijkstra can under certain conditions.

# MakeSet, Union, and Find

- Make-Set($i$): create a new set represented by $i$.
- Find($e$): name of the set $i$ that contains $e$ .
- Union($i, j$): union of the sets with names $i$ and $j$.

In MST-Kruskal:

- Make-Set($i$): New tree with $i$ as root.
- Find($e$): Find root of $e$
- Union($i, j$): Union of the trees $i$ and $j$.

# Algorithm MST-Kruskal($G$)

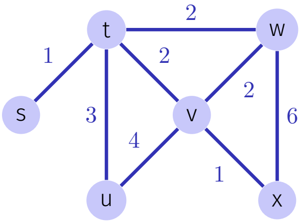
**Input:** Weighted Graph $G = (V, E, c)$
**Output:** Minimum spanning tree with edges $A$.

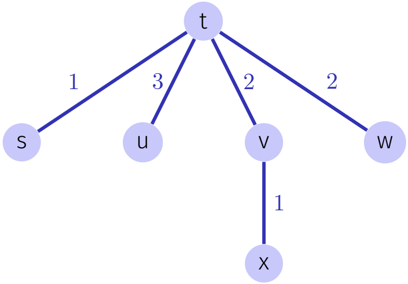
Sort edges by weight $c(e_1) \leq ... \leq c(e_m)$
$A \leftarrow \emptyset$
**for** $k = 1$ **to** $|V|$ **do**
    MakeSet($k$)

**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** $(V, A, c)$

# Representation as array




| Index | $s$ | $t$ | $w$ | $v$ | $u$ | $x$ |
|---|---|---|---|---|---|---|

| Index | $s$ | $t$ | $u$ | $v$ | $w$ | $x$ |
|---|---|---|---|---|---|---|
| Parent | $t$ | $t$ | $t$ | $t$ | $t$ | $v$ |

# Different kind of improvement

Link all nodes to the root when Find is called.

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** $i$

Cost: amortised *nearly* constant (inverse of the Ackermann-function).[2]

[2]We do not go into details here.

# Running time of Kruskal's Algorithm

- Sorting of the edges: $\Theta(|E| \log |E|) = \Theta(|E| \log |V|)$. [3]
- Initialisation of the Union-Find data structure $\Theta(|V|)$
- $|E|\times$ Union(Find($x$),Find($y$)): $\mathcal{O}(|E| \log |E|) = \mathcal{O}(|E| \log |V|)$.

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

[3] because $G$ is connected: $|V| \leq |E| \leq |V|^2$

# 7. Code-Expert Exercise

# Code-Example

’Kruskal MST’ on Code-Expert

# 8. TSP

# Travelling Salesperson Problem

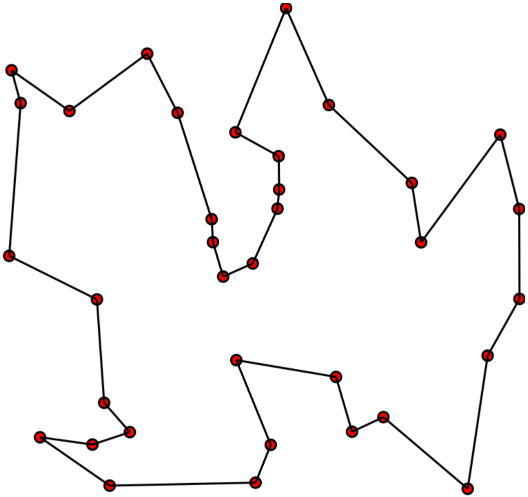
**Problem**

Given a map and list of cities, what is the shortest possible route that visits each city once and returns at the original city?

**Mathematical model**

On an undirected, weighted graph $G$, which cycle containing all of $G$'s vertices has the lowest weight sum?

# Travelling Salesperson Problem



# Travelling Salesperson Problem

- The problem has no known polynomial-time solution.
- Many heuristic algorithms exists. They do not always return the optimal solution.

# Travelling Salesperson Problem

- The heuristic algorithm that you are asked to implement on CodeExpert (*The Travelling Student*) on CodeExpert uses an MST:
  1. Compute the minimum spanning tree $M$
  2. Make a depth first search on $M$
- The algorithm is 2-approximate, meaning that the solution it generates has at most twice the cost of the optimal solution.
- The algorithm assumes a complete graph $G = (V, E, c)$that satisfies the triangle inequality: $c(v, w) \leq c(v, x) + c(x, w) \; \forall \; v, w, x \in V$

# 9. Old Exam Question

# Minimum Spanning Trees

### Aufgabe 3: Minimum Spanning Trees (MST) (14P)

Sie haben den Auftrag erhalten, den Übersetzungsprozess für eine Dokumentation zu leiten. Leider wurden verschiedene Abschnitte der Dokumentation in verschiedenen Sprachen verfasst, so dass es insgesamt $n$ verschiedene Sprachen gibt. Ihr Chef möchte, dass die gesamte Dokumentation in allen $n$ Sprachen verfügbar ist.
Es stehen $m$ verschiedene Übersetzer zur Verfügung, von denen jeder genau zwei der $n$ Sprachen beherrscht und zwischen ihnen hin- und herübersetzen kann. Jeder Übersetzer hat strikt positive Einstellungskosten. Leider haben Sie nicht genug Geld, um für jedes Sprachenpaar einen Übersetzer einzustellen. Stattdessen müssen Sie sich auf Übersetzerketten verlassen: Wenn Sie einen Englisch-Französisch-Übersetzer und einen Französisch-Spanisch-Übersetzer einstellen, können Sie die Dokumentation auch zwischen Englisch und Spanisch übersetzen. Ihr Ziel ist es, eine Gruppe von Übersetzern mit minimalen Kosten zu finden, die eine Übersetzung zwischen allen $n$ Sprachen ermöglicht.

*You have been hired to manage the translation process for some documentation. Unfortunately, different sections of the documentation were written in different languages, and there are $n$ different languages in total. Your boss wants the entire documentation to be available in all $n$ languages.*
*There are $m$ different translators for hire, and each of them knows exactly two of the $n$ languages and can translate back and forth between them. Each translator has a strictly positive hiring cost. Unfortunately, you don't have enough money to hire one translator for each pair of languages. Instead, you'll need to rely on chains of translators: if you hire an English-French translator and a French-Spanish translator, you'll also be able to translate the documentation between English and Spanish. Your goal is to find a set of translators of minimal cost that allows for translation between all $n$ languages.*

(a) Modellieren Sie das Problem als ein Problem des minimalen Spannbaums. Beschreiben Sie den Graphen (d.h. die Menge der Knoten, die Menge der Kanten und die Gewichte) in Worten.

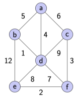
*Model the problem as a minimum spanning tree problem. Describe the graph (i.e. the set of vertices, the set of edges and the weights) in words.*

(b) Nennen Sie einen Algorithmus aus der Vorlesung zur Lösung des Problems des minimalen Spannbaums. Nennen Sie den Namen des Algorithmus und seine Laufzeit als Funktion von Anzahl Knoten $n$ und Anzahl Kanten $m$ in $\Theta$-Notation.

*Give an algorithm from the lecture to solve the minimum spanning tree problem. State the name of the algorithm and its running time in terms of number of nodes $n$ and number of edges $m$ in $\Theta$-notation.*

(c) Führen Sie den oben gewählten Algorithmus auf dem folgenden Graphen aus. Geben Sie den Namen des Algorithmus erneut an und geben Sie bei jedem Schritt an, welche Kante zum MST hinzugefügt wird.

*Run the algorithm selected above on the following graph instance. Give the name of the algorithm again and indicate at each step which edge you are adding to the MST.*




(d) Nehmen wir an, dass Ihr Chef zusätzlich zu den $n$ Sprachen die gesamte Dokumentation auch auf Schwiizerdütsch übersetzt haben möchte (was nicht zu den $n$ Originalsprachen gehörte). Es stehen $n$ zusätzliche Übersetzer zur Verfügung (die zwischen Schwiizerdütsch und jeder der $n$ Originalsprachen übersetzen), von denen jeder unterschiedliche positive Einstellungskosten hat. Sie hatten bereits den minimalen Spannbaum für die $n$ Originalsprachen berechnet, als Ihr Chef diese zusätzliche Anforderung stellte, also müssen Sie nun den minimalen Spannbaum für die $n+1$ Sprachen finden. Können Sie einfach den günstigsten Schwiizerdütsch-Übersetzer zum ursprünglichen Spannbaum hinzufügen? Begründen Sie Ihre Antwort.

*Suppose that additionally to the $n$ languages, your boss also wants the whole documentation translated in Swiss-German (which was not part of the $n$ original languages). You have at your disposal $n$ additional translators (translating between Swiss-German and each of the $n$ original languages), each of them having a different positive hiring cost. You had already computed the minimum spanning tree of the $n$ original languages when your boss asked for this extra requirement, so you now need to find the minimum spanning tree of the $n+1$ languages. Can you simply add the cheapest Swiss-German translator to your original spanning tree ? Justify your answer.*

# Minimum Spanning Trees – Solution

### Aufgabe 3: Minimum Spanning Trees (MST) (14P)

Sie haben den Auftrag erhalten, den Übersetzungsprozess für eine Dokumentation zu leiten. Leider wurden verschiedene Abschnitte der Dokumentation in verschiedenen Sprachen verfasst, so dass es insgesamt $n$ verschiedene Sprachen gibt. Ihr Chef möchte, dass die gesamte Dokumentation in allen $n$ Sprachen verfügbar ist.
Es stehen $m$ verschiedene Übersetzer zur Verfügung, von denen jeder genau zwei der $n$ Sprachen beherrscht und zwischen ihnen hin- und herübersetzen kann. Jeder Übersetzer hat strikt positive Einstellungskosten. Leider haben Sie nicht genug Geld, um für jedes Sprachenpaar einen Übersetzer einzustellen. Stattdessen müssen Sie sich auf Übersetzerketten verlassen: Wenn Sie einen Englisch-Französisch-Übersetzer und einen Französisch-Spanisch-Übersetzer einstellen, können Sie die Dokumentation auch zwischen Englisch und Spanisch übersetzen. Ihr Ziel ist es, eine Gruppe von Übersetzern mit minimalen Kosten zu finden, die eine Übersetzung zwischen allen $n$ Sprachen ermöglicht.

*You have been hired to manage the translation process for some documentation. Unfortunately, different sections of the documentation were written in different languages, and there are $n$ different languages in total. Your boss wants the entire documentation to be available in all $n$ languages.*
*There are $m$ different translators for hire, and each of them knows exactly two of the $n$ languages and can translate back and forth between them. Each translator has a strictly positive hiring cost. Unfortunately, you don't have enough money to hire one translator for each pair of languages. Instead, you'll need to rely on chains of translators: if you hire an English-French translator and a French-Spanish translator, you'll also be able to translate the documentation between English and Spanish. Your goal is to find a set of translators of minimal cost that allows for translation between all $n$ languages.*

(a) Modellieren Sie das Problem als ein Problem des minimalen Spannbaums. Beschreiben Sie den Graphen (d.h. die Menge der Knoten, die Menge der Kanten und die Gewichte) in Worten.

*Model the problem as a minimum spanning tree problem. Describe the graph (i.e. the set of vertices, the set of edges and the weights) in words.*

We can model the problem as an MST problem on an undirected graph whose vertices are given by the $n$ languages, and two vertices are connected by an edge if there is a translator knowing the two corresponding languages. The weight of an edge is the hiring cost of the corresponding translator.

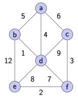
(b) Nennen Sie einen Algorithmus aus der Vorlesung zur Lösung des Problems des minimalen Spannbaums. Nennen Sie den Namen des Algorithmus und seine Laufzeit als Funktion von Anzahl Knoten $n$ und Anzahl Kanten $m$ in $\Theta$-Notation.

*Give an algorithm from the lecture to solve the minimum spanning tree problem. State the name of the algorithm and its running time in terms of number of nodes $n$ and number of edges $m$ in $\Theta$-notation.*

Kruskal's algorithm (runtime: $\Theta(m \log n)$).
Prim's algorithm (runtime: $\Theta(m + n \log n)$).

(c) Führen Sie den oben gewählten Algorithmus auf dem folgenden Graphen aus. Geben Sie den Namen des Algorithmus erneut an und geben Sie bei jedem Schritt an, welche Kante zum MST hinzugefügt wird.

*Run the algorithm selected above on the following graph instance. Give the name of the algorithm again and indicate at each step which edge you are adding to the MST.*

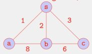



Here are the order in which the edges are added to the (unique) MST for the different algorithms.
Kruskal's algorithm: $bd$, $ef$, $cf$, $ad$, $ac$.
Prim's algorithm (started at vertex $a$): $ad$, $bd$, $ac$, $cf$, $ef$.
Prim's algorithm (started at vertex $b$ or $d$): $bd$, $ad$, $ac$, $cf$, $ef$.
Prim's algorithm (started at vertex $c$): $cf$, $ef$, $ac$, $ad$, $bd$.
Prim's algorithm (started at vertex $e$ or $f$): $ef$, $cf$, $ac$, $ad$, $bd$.

(d) Nehmen wir an, dass Ihr Chef zusätzlich zu den $n$ Sprachen die gesamte Dokumentation auch auf Schwiizerdütsch übersetzt haben möchte (was nicht zu den $n$ Originalsprachen gehörte). Es stehen $n$ zusätzliche Übersetzer zur Verfügung (die zwischen Schwiizerdütsch und jeder der $n$ Originalsprachen übersetzen), von denen jeder unterschiedliche positive Einstellungskosten hat. Sie hatten bereits den minimalen Spannbaum für die $n$ Originalsprachen berechnet, als Ihr Chef diese zusätzliche Anforderung stellte, also müssen Sie nun den minimalen Spannbaum für die $n+1$ Sprachen finden. Können Sie einfach den günstigsten Schwiizerdütsch-Übersetzer zum ursprünglichen Spannbaum hinzufügen? Begründen Sie Ihre Antwort.

*Suppose that additionally to the $n$ languages, your boss also wants the whole documentation translated in Swiss-German (which was not part of the $n$ original languages). You have at your disposal $n$ additional translators (translating between Swiss-German and each of the $n$ original languages), each of them having a different positive hiring cost. You had already computed the minimum spanning tree of the $n$ original languages when your boss asked for this extra requirement, so you now need to find the minimum spanning tree of the $n+1$ languages. Can you simply add the cheapest Swiss-German translator to your original spanning tree ? Justify your answer.*

No, this doesn't work. For example, if all the Swiss-German translators are cheaper than all the other translators, then the minimum spanning tree consist of all the Swiss-German translators, and does not contain any edge of the previously computed MST. (See below for a visual example, where $s$ stands for Swiss-German. The original MST consists of the edges $ab$ and $bc$, while the new spanning tree, after adding the node $s$, consists of the edges $as$, $bs$ and $cs$.)



# 10. Outro

# General Questions?

# Don’t forget!

**Next Week’s Session**

- Wed 8th May
- 16 - 18
- CHN G 42

# See you next time

Have a nice week!