4. Algorithms on graphs:

- How to know if a graph is connected?
- How to find the shortest path between two vertices?

  Def: A path is a vertex-simple chain.
- How to get out of a labyrinth "efficiently"?
- How to minimize your time in the u-bahn between Home, FU, Kneipe, Sports Club, Park and Friends?

Def: Let $G=(V,E)$ be a graph. A subgraph G' "(V',E')" of G is a spanning forest if $V'=V$ and G' is a forest.
(that is every vertex is in G' and it does not contain cycles).
If G' is connected, it is called a spanning tree.

Lemma: Let G be a connected graph. There exists a spanning tree G' of G.

We prove the lemma using an algorithm.

Algorithm: (Spanning Tree)
Input: Any graph G of order n with m edges.
Output: A subgraph G' of G.

Step 1: Order $E=(e_1, e_2, \dots, e_m)$

Step 2: Set $E_0' := \emptyset$.

Step 3: a) If E_{i-1}' was found, define

$$E_i' := \begin{cases} E_{i-1}' \cup \{e_i\} & \text{if } (V, E_{i-1} \cup \{e_i\}) \text{ has no cycle.} \\ E_{i-1} & \text{otherwise} \end{cases}$$

b) If $|E_i'| < n-1$ and $i < m$ repeat a)

Step 4: Let $G' = (V, E_t')$, where E_t' is the last set of edges in step 3.

Proposition about algorithm:

- If G' has $n-1$ edges then G' is a spanning tree of G.
- If G' has $k < n-1$ edges then G is a disconnected graph with $n-k$ components.

Pf] • E_i' do not contain cycles, if also G' has $n-1$ edges it is a tree. ✓

• If G' has $k < n-1$ edges it is a forest. Each component is a tree.

$\leadsto$ Each component contributes a "-1" to the number of vertices of the graph G'. $\Rightarrow$ $n-k$ components

$$\# \text{Edges} = n - \# \text{components}$$

$$\Leftrightarrow \# \text{components} = n - \# \text{edges}.$$

G' has $n-K$ components.

Left to prove: the vertex sets of the components of G' agree with the vertex sets of the components of G.
(is equal)

Assume the opposite, i.e. x and y connected in G but not in G'.

Let $C = (x_0, x_1), (x_1, x_2), (x_2, x_3) \dots (x_{l-1}, x_l)$ $\quad x_0 = x$, $x_l = y$
be a chain connecting x to y in G.

Let K be the component of G' containing x and x_i be the last vertex of K in C.
$\Rightarrow$ $i < l$ and so $x_{i+1} \notin K$.

$\Rightarrow$ $(x_i, x_{i+1}) \notin E'_*$
$\overset{\text{by algorithm}}{\Rightarrow}$ $e = (x_i, x_{i+1})$ had to form a cycle inside G'
$\Rightarrow$ $G' \cup \{e\}$ also contains a cycle, but e connects two connected comp. of G'. $\lightning$ $\boxtimes$

This algorithm adds edges according to a specific ordering of the edges.
$\leadsto$ But at every step, we have to check if we created a cycle.

Algorithm: (Growing a spanning tree from a root).

Input: - A graph $G=(V,E)$ of order n with m edges.
- A vertex $v \in V$.

Output: A subgraph G' of G.

Step 1: Set $E'_0 = \emptyset$ and $V'_0 = \{v\}$.

Step 2: a) Given E'_{i-1} and V'_{i-1}, find $e_i = \{x_i, y_i\} \in E$ such that $x_i \in V'_{i-1}$ and $y_i \in V \setminus V'_{i-1}$

b) Set $E'_i := E'_{i-1} \cup \{e_i\}$
$V'_i := V'_{i-1} \cup \{y_i\}$.

c) If an edge was found, repeat a) followed by b).

Step 3: Obtain $G' = (V'_*, E'_*)$ E'_* and V'_* are the last step's sets.

Proposition about algorithm:

If G' has n vertices, then G' is a spanning tree.
Else G is a disconnected graph and G' is a spanning tree of the component containing v.

Pf) G' with n vertices and $n-1$ edge which is connected.
$\Rightarrow$ G' is a tree.

If G' has $n' < n$ vertices, we show that the component of G containing v has n' vertices.

By contradiction, assume $\exists\, y \in V$ and $x \in V'$ connected by a path in G to v.

$\Rightarrow \exists$ edge (u,w) in this path from x to y s.t.

$u \in V'$ and $w \in V \setminus V'$.

At that point, (u,w) is a valid edge for Step 2a). Hence it would have added it before stopping. $\lightning$ $\boxtimes$.

We now have two algorithms to get spanning trees. How many possible outputs are there for the first algorithm on the complete graph on "n" vertices?

Equivalently how many labeled trees on "n" vertices? $\underline{\underline{n^{n-2}}}$.

<u>Thm</u>: (Cayley)

The number of labeled trees on n vertices is n^{n-2}.

<u>Proof</u>: (Prüfer, 1918)

<u>Idea</u>: Use algorithm on spanning trees.

A) Backtrack Algorithm # 2:

- Take a spanning tree and remove its smallest leaf l_1 and the edge incident to it.
- Record in p_1 the neighbor of l_1.
- Proceed $n-2$ times until you get only two vertices and 1 edge.

↳ End with $p = p_1 p_2 \cdots p_{n-2}$.

B) Reconstruct a tree from a sequence:

- Given $p = p_1 p_2 \cdots p_{n-2} \in [n]^{n-2}$, construct a spanning tree:

Facts:
- p_i's cannot be leaves
- Any vertex in $[n] \setminus \{p_1, p_2, \dots, p_{n-2}\}$ is a leaf.

- By the "removing leaves" action l_1 has to be $\min [n] \setminus \{p_1, p_2, \dots, p_{n-2}\}$ so we connect it to p_1.

$$l_1 \bullet\!\!-\!\!\!-\!\!\bullet\, p_1$$

Then $l_2 \neq l_1$ and $l_2 \notin \{p_2, \dots, p_{n-2}\} \Rightarrow l_2 = \min [n] \setminus \{p_2, \dots, p_{n-2}, l_1\}$

$\rightsquigarrow l_i = \min [n] \setminus \{p_i, \dots, p_{n-2}, l_1, l_2, \dots, l_{i-1}\}$.

- Remains the edge $\{p_{n-2}, [n]\setminus\{l_i\}\cup\{p_{n-2}\}\}$.
- To finish the proof, show that the subgraph is a tree
- and that applying the part A) on the subgraph gives back p.

(Excercise). □

Breadth-first and Depth-first search

Two types of strategies can be used to produce spanning trees:

Breadth-first search:

Step 1) Pick $v \in V$ to be the current vertex, and label it by 1.

Step 2) • Add all neighbors of the current vertex "i" to the spanning tree with "r" vertices.

• Label them by $r+1, r+2, \ldots$

• Add edges $\{i, r+1\}\{i, r+2\}, \ldots$

• If the number of vertices is now "n", stop. Else, make "$i+1$" the current vertex.

Depth-first search:

Step 1: Pick $v \in V$ to be the start vertex and label it "1". Stack: (1)

Step 2: • Say current vertex is s_t, stack $(s_1, s_2, \dots, s_t)$ and "r" vertices have been labeled:

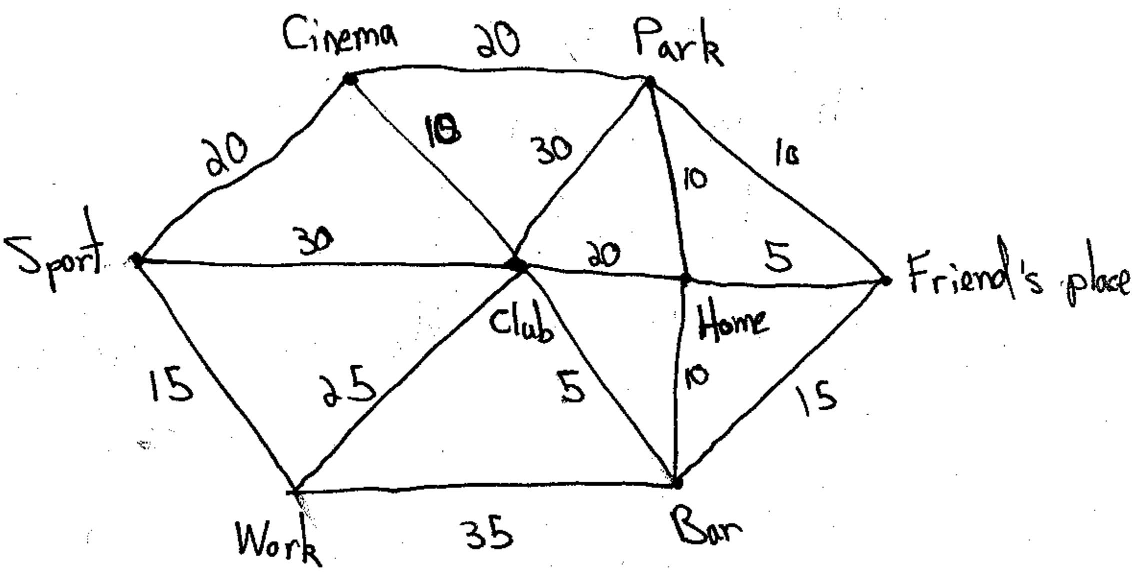
- Choose a not yet labeled neighbor of "s_t" and label it by $r+1$ and add $\{i, r+1\}$.

  Stack: $(s_1, s_2, \dots, s_t, r+1)$
  Current vertex: $r+1$

- If there are no unlabeled neighbor, go to top stack remove last entry and go to the now updated last entry.

- If stack empty, finish.

Minimal Spanning Tree:



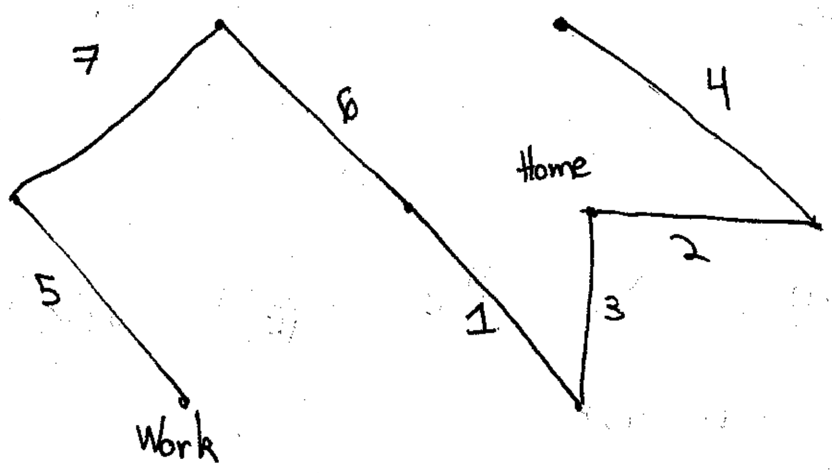
<u>Problem:</u> Find a network/subgraph such that globally, the time is minimized:

Say $\omega : E \to \mathbb{R}$

Find a spanning tree T such that

$$\sum_{e \in T} w(e) \text{ is minimal.}$$

<u>Naive/Greedy approach:</u> Always choose the edge with min. weight that does not create a cycle.



$\Rightarrow$ 65 mins to go to work! (45 min seems best...).

<u>Kruskal's Algorithm (or greedy algorithm):</u>

<u>Input:</u> $G=(V,E)$ connected graph.
$w : E \to \mathbb{R}$.

<u>Output:</u> G' spanning tree of G minimizing $\sum_{e \in G'} w(e)$.

Step 1) Order $E = \{e_1, \ldots, e_m\}$ such that

$$w(e_1) \leq w(e_2) \leq \cdots \leq w(e_m).$$

Step 2) Apply Algorithm "Spanning Tree".

Proof of correctness:

- Let T be the spanning tree found.
  and $\check{T}$ be any other spanning tree.
  To show $$\sum_{e \in T} w(e) \leq \sum_{e \in \check{T}} w(e).$$
- Order the edge $e'_1, e'_2, \ldots, e'_{n-1}$ so that $w(e'_1) \leq w(e'_2) \leq \cdots \leq w(e'_{n-1})$
  $\check{e}_1, \check{e}_2, \ldots, \check{e}_{n-1}$ so that $w(\check{e}_1) \leq \cdots \leq w(\check{e}_{n-1})$.

By contradiction, assume that $w(e'_i) > w(\check{e}_i)$ for some $1 \leq i \leq n-1$
(Stronger than necessary).

Let
$$E' = \{e'_1, e'_2, \ldots, e'_{i-1}\} \qquad |E'| = i-1$$
$$\check{E} = \{\check{e}_1, \ldots, \check{e}_i\} \quad \Rightarrow \quad |\check{E}| = i$$

Observe: (V, E') and $(V, \check{E})$ are forests.

We show that $\exists\, e \in \check{E}$ s.t. $(V, E' \cup \{e\})$ is a forest.

$\Rightarrow \quad w(e) \leq w(\check{e}_i) < w(e'_i)$.

$\Rightarrow$ The algorithm did a mistake, we should have selected e instead of e_i.

<u>Claim</u>: (smelling matroids coming).

If $E', \check{E} \subseteq \binom{V}{2}$ such that $(V, \check{E})$ is a forest and $|E'| < |\check{E}|$ then some $e \in \check{E}$ connects vertices of distinct components of the graph (V, E').

<u>Pf of claim:</u>

Let $V_1, V_2, \dots, V_s$ be the vertex sets of conn. comp. of (V, E')

$$|E' \cap \binom{V_j}{2}| \geq |V_j| - 1$$

By previous exercises.

Summing over j's $\Rightarrow$ $|E'| \geq n - s$ (Also exercise).

Because $\check{E}$ is a forest:

$$|\check{E} \cap \binom{V_j}{2}| \leq |V_j| - 1.$$

So $\check{E}$ has at most $n-s$ edges contained in the j conn. comp. of (V, E'). $\underset{|\check{E}| > |E'|}{\Rightarrow}$ $\exists\, e \in \check{E}$ between two conn. comp $\boxtimes$ claim.

Now, since E' was a forest, putting an edge between 2 conn. comp. yields a forest ↯ ☒

This is quite exceptional that <u>greedy = optimal</u>!!!!

Shortest Path Problem

Say that I would like to have shortest paths from my home to every place in the graph?

We use <u>Breadthfirst Seach</u>.

Dijkstra's Alogrithm:

<u>Input</u>: A connected graph $G=(V,E)$ and $w: E \to \mathbb{R}_{\geq 0}$ and $v \in V$.

<u>Output</u>: Spanning tree G' such that $\forall w \in V$, the path from v to w in G' has minimal weight among all chains in G from v to w.

<u>Idea</u>: Use Breadthfirst Search.

<u>Step 1)</u> • Assign $\delta^p(v) = 0$ permanent and temporarly $\delta^t(w) = \infty$ $\forall w \in V$, $w \neq V$.

• Set v to be the "current vertex."

<u>Step 2)</u> • Assume p is the current vertex

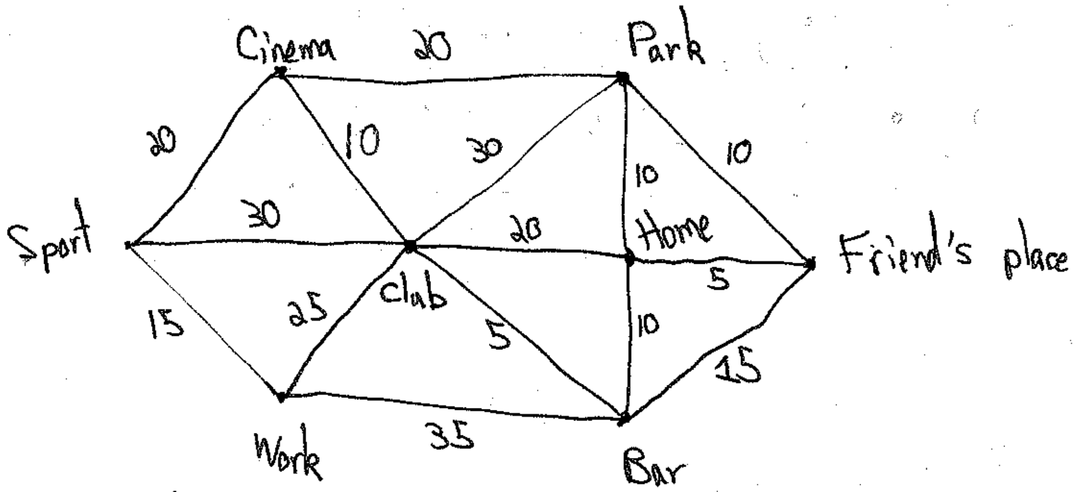
• For all neighbors "w" of p that have temporary δ's, reset $\delta^t(w) = \min\{\delta^t(w), \delta(p) + w(pw)\}$.

<u>Step 3</u> : • Take the vertex y with smallest temporary δ^t and make its δ permanent.
• Add the edge yq where q is the vertex used to the tree.
• Make y the "current vertex

<u>Step 4:</u> Repeat Step 2 and 3 if there are still temporary δ's.

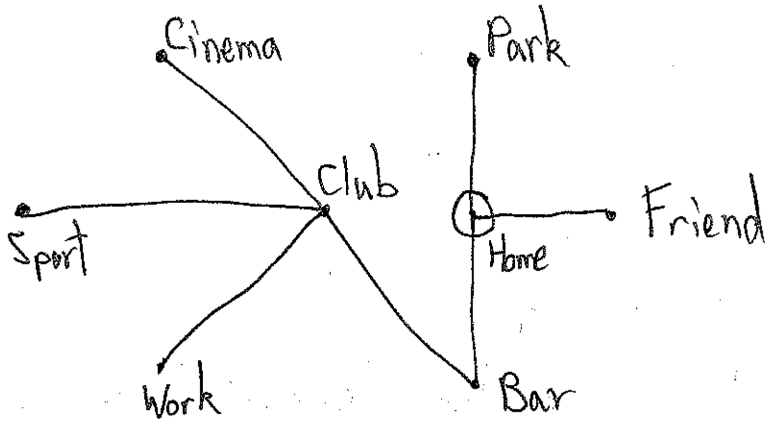
<u>pf of correctness:</u> Induction on number of visited vertices □

<u>Example:</u>

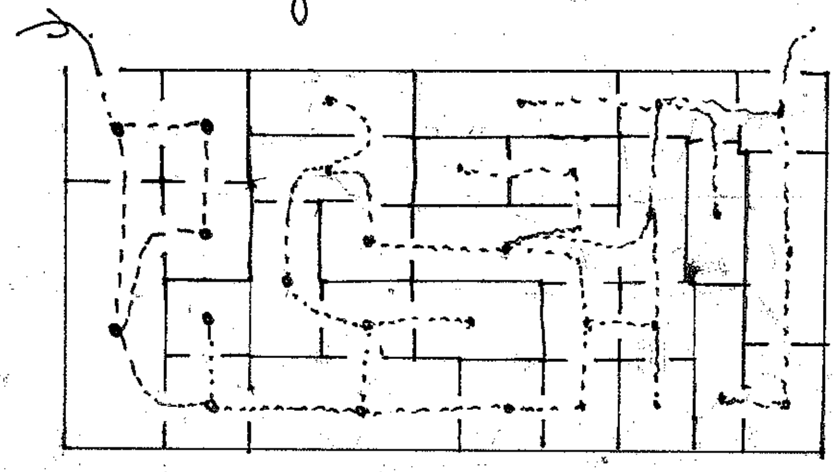


Say "Home" is root

Current vertex	Home	Park	Bar	Friend	Club	Cinema	Work	Sport	Y	New Ed
—	$\boxed{0}$	∞	∞	∞	∞	∞	∞	∞	—	—
Home		10	10	$\boxed{5}$	20	∞	∞	∞	Friend	Home/Fr
Friend		$\boxed{10}$	10		20	∞	∞	∞	Park	Home/Par
Park			$\boxed{10}$		20	30	∞	∞	Bar	Home/Ba
Bar					$\boxed{15}$	30	45	∞	Club	Bar/Clu
Club						$\boxed{25}$	40	45	Cinema	Club/Cin
Cinema							$\boxed{40}$	45	Work	Club/Wor
Work								$\boxed{45}$	Sport	Club/Spo



<u>Example</u>: How to get out of this maze "fast"?



Use <u>Depth-first Search</u>! That is: run through room like a panicking child!

But! Follow the rules:

1) Come back to "previous room" when

a) You have been in all doors or no doorsare there

b) You enter a room that you have seen.

Can you think of a "maze" where "always turn right" will not bring you to the exit?