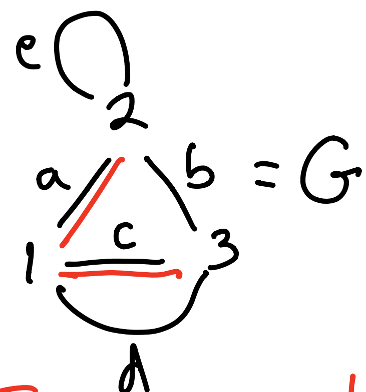
Math 4707 Nov. 4, 2020

THEOREM (Kirchhoff 1848 "Matrix-Tree Theorem")

For a graph $G=(V,E)$,

$$\underset{\substack{\| \\ \#\text{spanning} \\ \text{trees in } G}}{\tau(G)} = \det\Big(\underbrace{\overline{L(G)}^{v}}_{\text{reduced Laplacian matrix}}\Big) \text{ for any vertex } v\in V$$

$L(G)$ − v^{th} row, v^{th} column



$= G$

T a spanning tree in G

$$L(G) = \begin{array}{c c} & \begin{matrix} 1 & 2 & 3 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} & \begin{bmatrix} 3 & -1 & -2 \\ -1 & 2 & -1 \\ -2 & -1 & 3 \end{bmatrix} \end{array}$$

$$\overline{L(G)}^{3} = \begin{array}{c c} & \begin{matrix} 1 & 2 & 3 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} & \begin{bmatrix} 3 & -1 & \cancel{-2} \\ -1 & 2 & \cancel{-1} \\ \cancel{-2} & \cancel{-1} & \cancel{3} \end{bmatrix} \end{array}$$

$$\det \overline{L(G)}^{3} = \det\begin{bmatrix} 3 & -1 \\ -1 & 2 \end{bmatrix}$$

$$= 3\cdot 2 - (-1)(-1) = 5$$

$$= \tau(G)$$

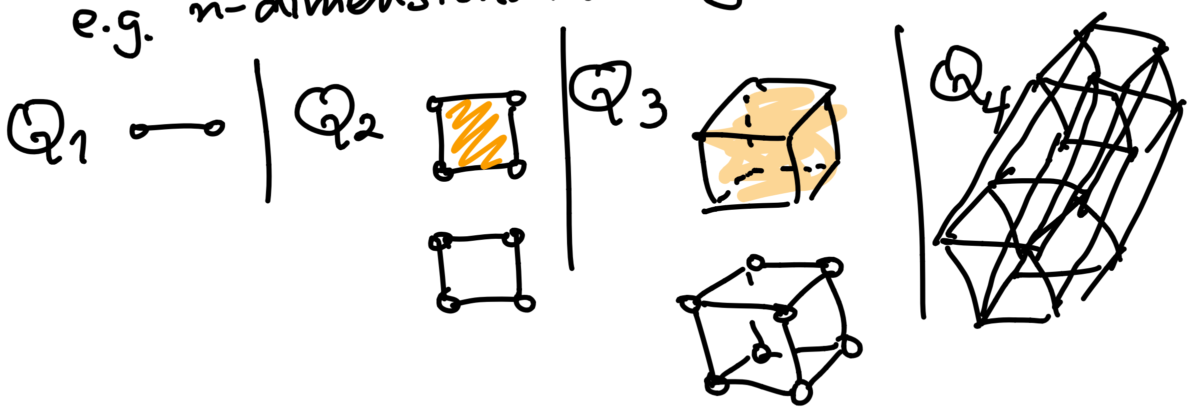
Theoretical example:

$$\underset{\substack{\| \\ n^{n-2}}}{\tau(K_n)} \quad \text{Borchardt-Cayley} \quad = \det\left(\overline{L(K_n)}^{v}\right) = \text{product of eigenvalues of } L(\widetilde{K_n})$$

$$= \underbrace{n\cdot n\cdot \cdots \cdot n}_{n-2}\cdot 1 = n^{n-2}$$

REMARK: There are many families of highly structured graphs G for which $\tau(G) = \det(L(G)^{v})$ = product of eigenvalues is the easiest way to compute $\tau(G)$.

e.g. n-dimensional cube graphs Q_n



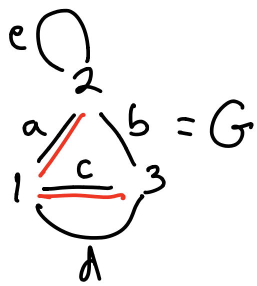
THEOREM (Kirchhoff 1848 "Matrix-Tree Theorem")

For a graph $G=(V,E)$,

$$\underset{\substack{\shortparallel \\ \text{\# spanning trees in } G}}{\tau(G)} = \det\Big(\underbrace{\overline{L(G)}^{\,v}}_{\text{reduced Laplacian matrix}}\Big) \quad \text{for any vertex } v\in V$$

reduced Laplacian matrix

$L(G) - v^{th}$ row, v^{th} column



$$L(G) = \begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline 1 & 3 & -1 & -2 \\ 2 & -1 & 2 & -1 \\ 3 & -2 & -1 & 3 \end{array}$$

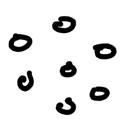
proof: Let's induct on $|E|$.

BASE CASE: $|E|=0$

SUBCASE (a): $|V|=1$ $\quad G = \overset{v}{\circ} \quad L(G) = v\,[\overset{v}{0}]$

$$\det \overline{L(G)}^{\,v} = \det\big(\cancel{v[0]}\big) = \det([\ ]) = 1 = \tau(G) \ \checkmark$$

SUBCASE (b): $|V| \geq 2$

$G =$  $\quad L(G) = \begin{bmatrix} 0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&0 \end{bmatrix}$

$$\det \overline{L(G)}^{\,v} = \det\left(\begin{bmatrix} 0&0&0&\cancel{0}\\0&0&0&\cancel{0}\\0&0&0&\cancel{0}\\ \cancel{0}&\cancel{0}&\cancel{0}&\cancel{0} \end{bmatrix}\right) = 0 = \tau(G)$$

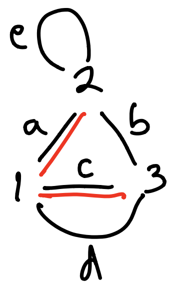
THEOREM (Kirchhoff 1848 "Matrix-Tree Theorem")

For a graph $G = (V, E)$,

$$\underset{\substack{\| \\ \#\text{spanning} \\ \text{trees in } G}}{\tau(G)} = \det\big(\underbrace{\overline{L(G)}^{v}}\big) \text{ for any vertex } v \in V$$

reduced Laplacian matrix

$L(G)$ − v^{th} row, v^{th} column



$= G$

$$L(G) = \begin{array}{c|c} & \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \end{array} & \begin{bmatrix} 3 & -1 & -2 \\ -1 & 2 & -1 \\ -2 & -1 & 3 \end{bmatrix} \end{array}$$

---

proof (continued):

INDUCTIVE STEP:

SUBCASE (a): $v \in V$ is isolated in G.

e.g.

$\bullet\, 4 = v$

has
$$L(G) = \begin{array}{c|c} & \begin{array}{cccc} 1 & 2 & 3 & 4{=}v \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ v{=}4 \end{array} & \begin{bmatrix} 3 & -1 & -2 & 0 \\ -1 & 2 & -1 & 0 \\ -2 & -1 & 3 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

$$\overline{L(G)}^{v} = L\left(G \text{ with vertex } v \text{ removed}\right)$$

$$\det \overline{L(G)}^{v} = \det \begin{bmatrix} 3 & -1 & -2 \\ -1 & 2 & -1 \\ -2 & -1 & 3 \end{bmatrix} = 0 = \tau(G)$$

since $\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$ is in the nullspace = kernel

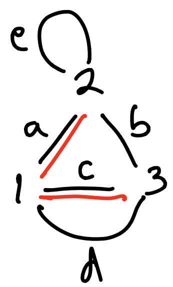
i.e. any row of $L(G)$ say corresponding to a vertex v_0 has entries summing to <u>zero</u>. (Think about it!)

THEOREM (Kirchhoff 1848 Matrix-Tree Theorem)

For a graph $G=(V,E)$,

$$\underbrace{\tau(G)}_{\substack{\#\text{spanning} \\ \text{trees in } G}} = \det\Big(\underbrace{\overline{L(G)}^{v}}_{\text{reduced Laplacian matrix}}\Big) \quad \text{for any vertex } v\in V$$

$\overline{L(G)}^{v}$: $L(G)$ − v^{th} row, v^{th} column



$= G \qquad L(G) = \begin{array}{c c} & \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \end{array} & \begin{bmatrix} 3 & -1 & -2 \\ -1 & 2 & -1 \\ -2 & -1 & 3 \end{bmatrix} \end{array}$

---

proof: INDUCTIVE STEP subcase (b):

Our vertex $v\in V$ is not isolated in G, say some edge $e=\{v,w\}$ exists in G.

Recall $\tau(G)=\tau(G\setminus e)+\tau(G/e)$

Strategy: Relate $\overline{L(G)}^{v}$, $\overline{L(G\setminus e)}^{v}$, $\overline{L(G/e)}^{vw}$,

to get $\boxed{\det \overline{L(G)}^{v} = \det \overline{L(G\setminus e)}^{v} + \det \overline{L(G/e)}^{vw}}$

and then we'd be done by induction on $|E|$, since

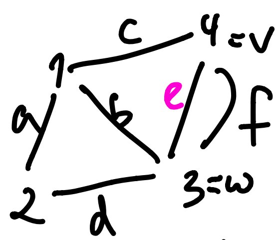
$$\tau(G) = \tau(G\setminus e) + \tau(G/e)$$

Why is

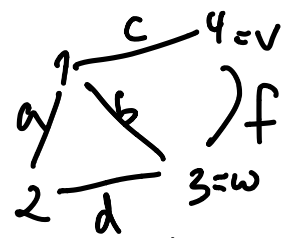
$$\boxed{\det \overline{L(G)}^{v} = \det \overline{L(G\setminus e)}^{v} + \det \overline{L(G/e)}^{vw}}\ ?$$

Proof by example!

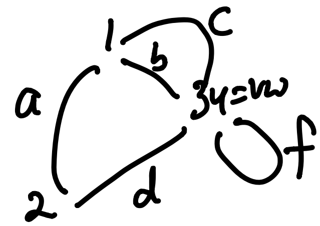
G



$G \setminus e$



G/e



$$\overline{L(G)}^{v} \qquad \begin{array}{c|cccc} & 1 & 2 & 3 = w & 4 = v \\ \hline 1 & 3 & -1 & -1 & -1 \\ 2 & -1 & 2 & -1 & 0 \\ w=3 & -1 & -1 & 4 & -2 \\ 4 = v & -1 & 0 & -2 & 3 \end{array}$$

$$\overline{L(G\setminus e)}^{v} \qquad \begin{array}{c|cccc} & 1 & 2 & 3 = w & 4 = v \\ \hline 1 & 3 & -1 & -1 & -1 \\ 2 & -1 & 2 & -1 & 0 \\ w=3 & -1 & -1 & 3 & -1 \\ v=4 & -1 & 0 & -1 & 2 \end{array}$$

$$\overline{L(G/e)}^{vw} \qquad \begin{array}{c|ccc} & 1 & 2 & 34 = vw \\ \hline 1 & 3 & -1 & -2 \\ 2 & -1 & 2 & -1 \\ vw = 34 & -2 & -1 & 3 \end{array}$$

and hence by computing $\det \overline{L(G)}^{v}$ by Laplace expansion along last row or column,

$$\det \overline{L(G)}^{v} = \det \overline{L(G\setminus e)}^{v} + \det \overline{L(G/e)}^{vw}$$

because of the difference by 1 in the green circled entries on the diagonal at (w,w). ∎

Minimum cost spanning trees

If the edges of the graph G are assigned costs $\mathrm{cost}(e) \geq 0$, how we can find a subset $T \subset E$ of edges that connect all the vertices $v \in V$ with minimum cost

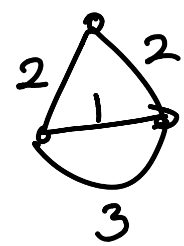
$$\mathrm{cost}(T) := \sum_{e \in T} \mathrm{cost}(e)$$

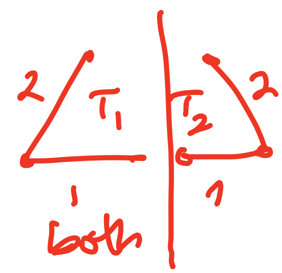
NOTE: T must be a spanning tree in G, but it might not be unique.

e.g.

$\mathrm{cost}(T) = 3 + 5 = 8$

e.g.





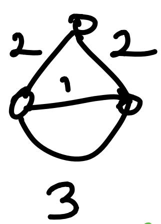
both have $\mathrm{cost}(T_1) = \mathrm{cost}(T_2) = 1 + 2 = 3$

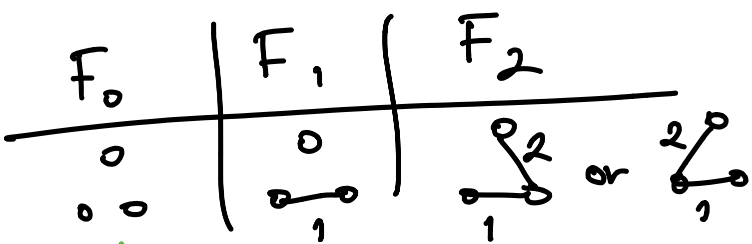
There is a surprisingly simple and fast <u>greedy</u> algorithm that always works:

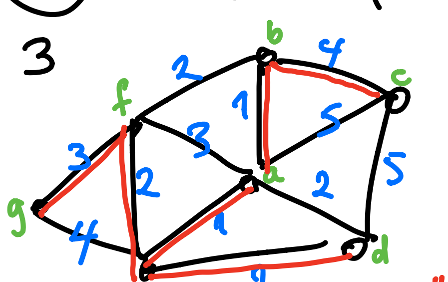
<u>Kruskal's greedy algorithm:</u>

- Start with an <u>empty</u> forest F_0 having <u>no edges</u>
- At each step, add the cheapest edge e to F_i so that $F_{i+1} = F_i \cup \{e\}$ is still a forest, that is, <u>no cycle is created</u>.
- Stop when $|F_i| = |V|-1$, that is, $i = |V|-1$.
  This gives $F_{|V|-1} =: T_{greedy}$ a spanning tree.

EXAMPLE:

(1) $G=$ 



(2) 

cost$(T_{greedy}) = 3+2+1+1+1+4$
$=12$

edge costs in blue

Math 4707 Nov. 9, 2020

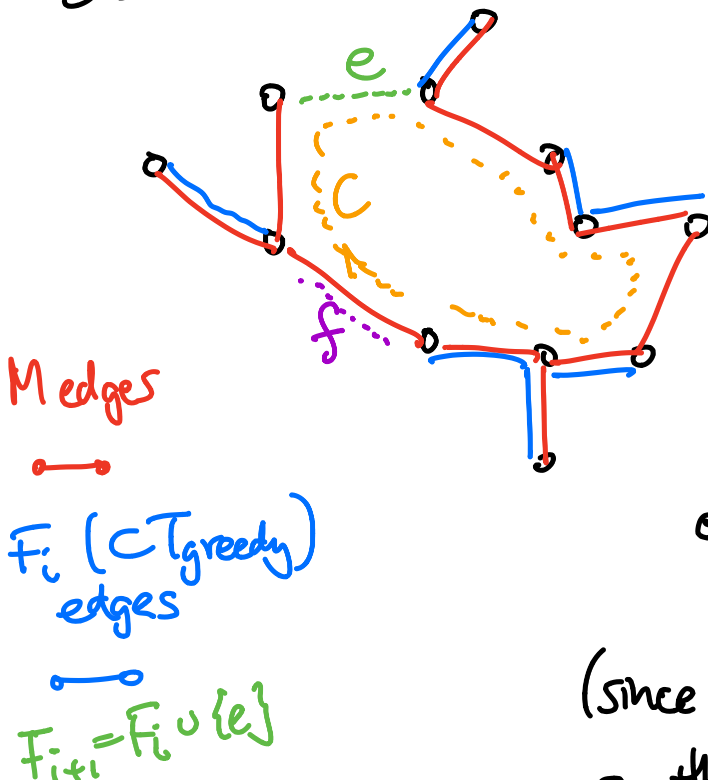
THEOREM: Kruskal's algorithm for finding a minimum cost spanning tree T in $G=(V,E)$ (greedily add in cheapest edge that doesn't create a cycle at each step) always succeeds in finding a minimum-cost spanning tree.

proof: Let T_{greedy} be (the, any) tree produced by Kruskal's algorithm from $G=(V,E)$ with edge costs $cost(e)$.

Let M be any minimum cost spanning tree. We'll show $M = T_{greedy}$.

If $M \neq T_{greedy}$, then find the earliest stage during Kruskal's algorithm where we we pick an edge e to add to F_i (in the i^{th} stage of building T_{greedy}) to create $F_{i+1} = F \cup \{e\}$ but $e \notin M$.

So this means $F_i \subset M \cap T_{greedy}$.

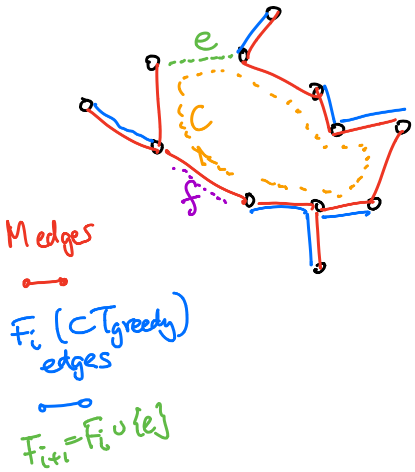


Then adding e to M creates a uniqe cycle C in $M \cup \{e\}$.

Furthermore $C - \{e\}$ must contain an edge f of M <u>not</u> in T_{greedy}.

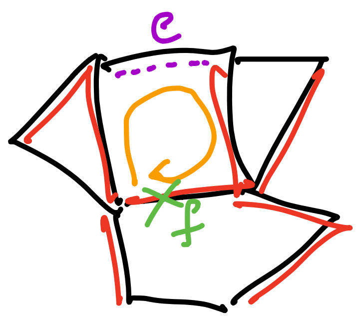
(since $e \in F_{i+1} \subset T_{greedy}$, so otherwise $T_{greedy} \supset C$ <u>not acyclic</u>)

Now let $M' := (M - \{f\}) \cup \{e\}$



Now let $M' := (M - \{f\}) \cup \{e\}$

This M' is still a spanning tree for G (Why?)



Think about this a bit:
- Why acyclic?
- Why connecting all vertices?

Also $\text{cost}(f) \geq \text{cost}(e)$ since f can't form a cycle with F_i because

$$\underbrace{F_i}_{\subseteq M} \cup \{f\} \subset M \text{ a spanning } \underline{\text{tree}}.$$

So f competed with e at i^{th} stage, and $\text{cost}(e) \leq \text{cost}(f)$.

Thus
$$\begin{aligned} \text{cost}(M') &= \text{cost}(M) - \text{cost}(f) + \text{cost}(e) \\ &= \text{cost}(M) - (\underbrace{\text{cost}(f) - \text{cost}(e)}_{\geq 0}) \\ &\leq \text{cost}(M). \end{aligned}$$

Hence $\text{cost}(M') = \text{cost}(M)$ since $\text{cost}(M)$ minimum.

And M' agrees with T_{greedy} in 1 more edge. $\blacksquare$

§9.2 Traveling Salesperson Problem (TSP)

This is a common optimization problem where we're given n points (vertices) and costs/distances/travel times $d(v,v')$ for every pair v, v' of vertices.

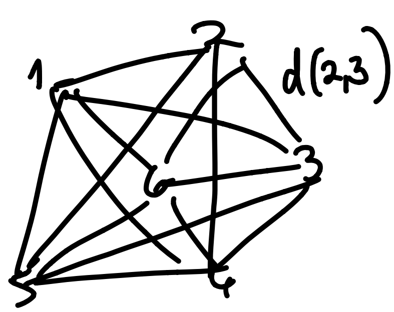
We want to find a closed tour C

$$v_1 - v_2 - v_3 - \dots - v_n - \underset{v_{n+1}}{v_1}$$

visiting every vertex exactly once having $d(C) := \sum_{i=1}^{n} d(v_i, v_{i+1})$ as small as possible.

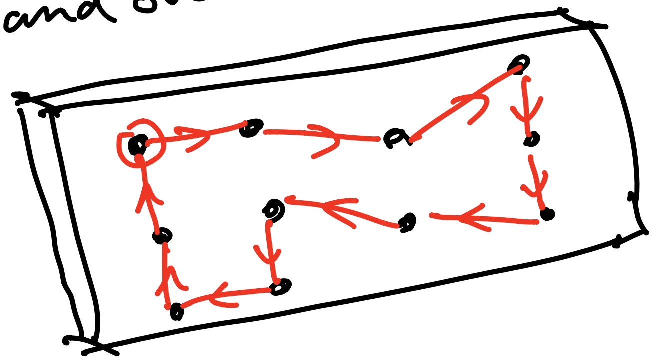
EXAMPLES:

(1) Traveling salesperson needs to visit these towns in some order and return home:

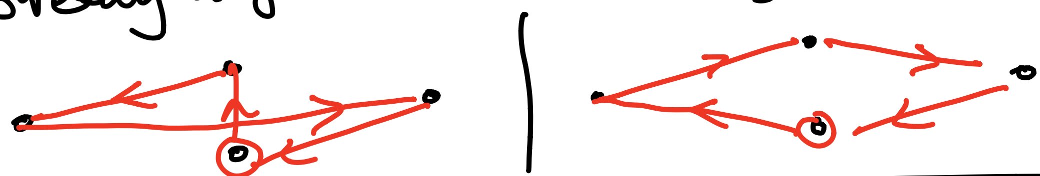


$d(v,v')$ might be
- travel time
- travel cost
- distance

(2) (From book) An industrial drill must make holes at certain positions on a circuit board, over and over



Greedy algorithms can easily fail, e.g.



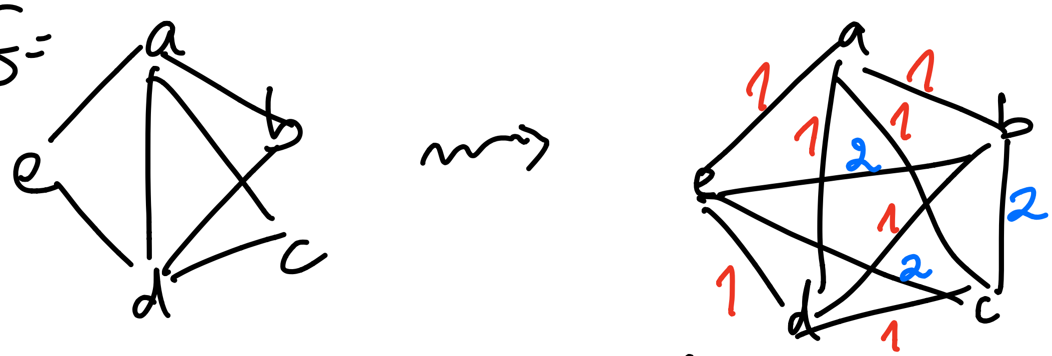
Q: Are there fast algorithms (like Kruskal for minimum cost spanning tree) for solving large instances of TSP's ?

Probably not ! If we had such an algorithm it would give a fast algorithm to decide if a graph $G = (V, E)$ has a Hamiltonian cycle (or many other problems)...

Given $G = (V, E)$, put distances

$$d(v,v') = \begin{cases} 1 & \text{if } \{v,v'\} \in E \\ 2 & \text{if } \{v,v'\} \notin E \end{cases}$$

$G =$



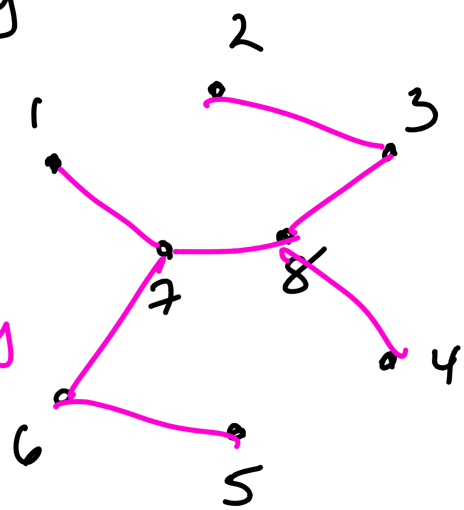
<u>CLAIM</u>: G has a Hamiltonian cycle $\iff$ the graph on right with edge costs given has minimum cost TSP tour of cost $|V|$.

Instead, people started looking for <u>fast</u> approximation algorithms, not necessarily finding optimum, but guaranteed <u>not too far</u>.

The "tree short-cutting" algorithm for TSP:

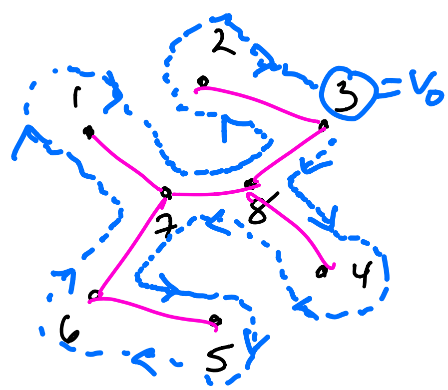
① Use the cost/distances $d(v,v')$ to find via Kruskal a minimum cost spanning tree T_{greedy} connecting the points

e.g.



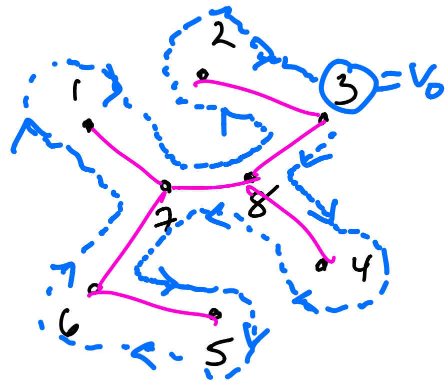
T_{greedy}

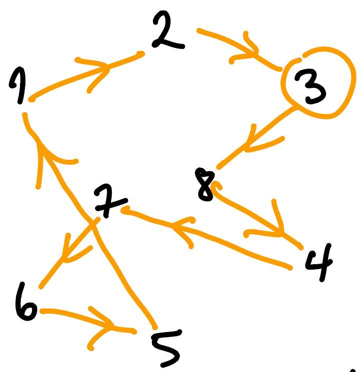
② Starting at any vertex v_0 as root, walk around the perimeter, going over each edge twice, and return to v_0, creating a tour $C_{walkaround}$



$C_{walkaround}$

③ Create $C_{shortcut}$ by taking "shortcuts" past the vertices that you already visited.



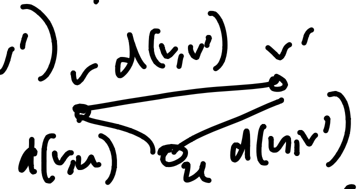


$C_{shortcut}$ IS OUR APPROXIMATION!

THEOREM: The tour $C_{shortcut}$ produced by tree shortcutting algorithm has

$$d(C_{shortcut}) \le 2\, d(C_{optimal})$$

under the assumption that $d(v,v')$ satisfy the triangle inequality:

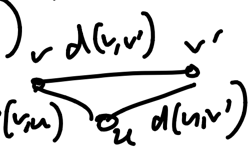


$$d(v,v') \le d(v,u) + d(u,v') \quad \forall\, u,v,v' \in V$$

THEOREM: The tour $C_{shortcut}$ produced by tree shortcutting algorithm has

$$d(C_{shortcut}) \le 2\, d(C_{optimal})$$

under the assumption that $d(v,v')$ satisfy the triangle inequality:



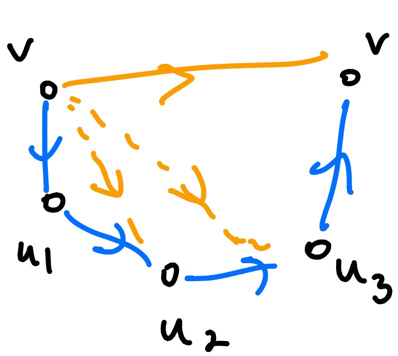
$$d(v,v') \le d(v,u) + d(u,v') \quad \forall\, u,v,v' \in V$$

proof:

$$d(C_{shortcut}) \le d(C_{walkaround}) = 2\, d(T_{greedy})$$

need triangle inequality here!

on the nose!



$$\le 2\, d(\underbrace{C_{optimal} - \{\text{any edge}\}}_{\text{because this is a spanning tree}})$$

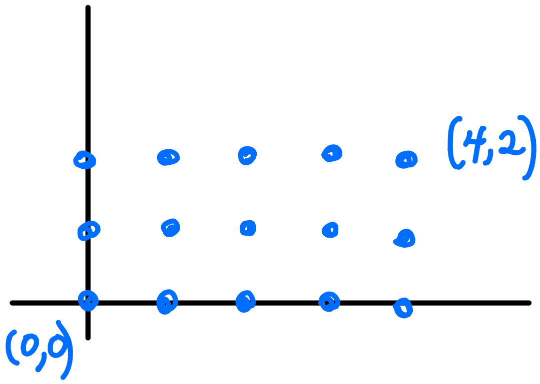
$$\le 2\, d(C_{optimal})$$

∎

Math 4707 Nov. 11, 2020

Practice with

Tree shortcutting algorithm for Traveling Salesperson Problem



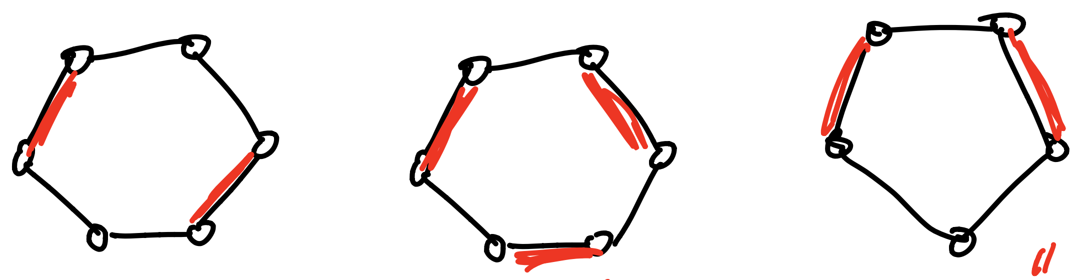
① For the 15 points above, using Euclidean distance $d(u,v)$ as the cost for traveling between u and v, find a TSP tour $C_{shortcut}$ via the tree-shortcutting algorithm.

② Find an optimal (shortest) $C_{optimal}$ for these points, and prove its optimality.

③ What was $\dfrac{d(C_{shortcut})}{d(C_{optimal})}$ for your choice?

Chapter 10 Matching Theory

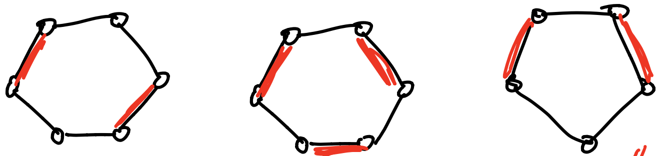
<u>DEF'N</u>: A <u>matching</u> M in a graph $G=(V,E)$ is a subset $M \subset E$ of the edges that share no vertices. M is called <u>maximum</u> (-<u>sized</u>) if $\nexists$ a matching $M' \subset E$ with $\#M' > \#M$. It's called a <u>perfect matching</u> if every vertex $v \in V$ is matched in M, that is, every v is an endpoint of some $e \in M$. This requires $\#M = \frac{\#V}{2}$ so $\#V$ even



M — a matching, not maximum, not perfect

M' — a perfect matching, and maximum

M'' — a maximum matching, not perfect



M — a matching, not maximum, not perfect

M' — a perfect matching, and maximum

M'' — a maximum matching, not perfect

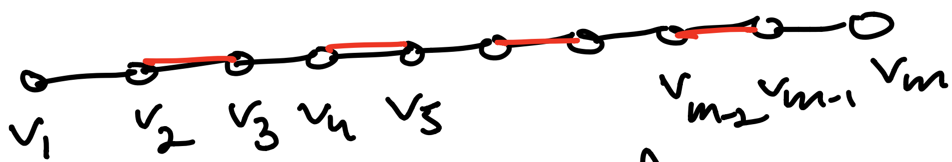
Q: How to find quickly maximum matchings M in G, particularly if G is large?

A: We have good algorithms for any graph G, particularly fast and simple for bipartite graphs.

Not as fast or easy as Kruskal for spanning trees, i.e. not greedy.

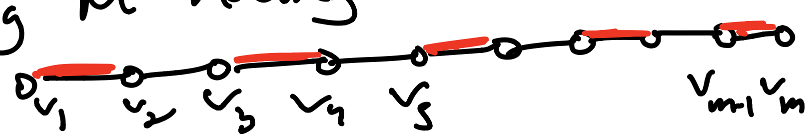
This local improvement in path turns out to be the key for the algorithms.

DEF'N: If M is a matching in $G=(V,E)$ then an **M-augmenting path** P in G is a path like this:

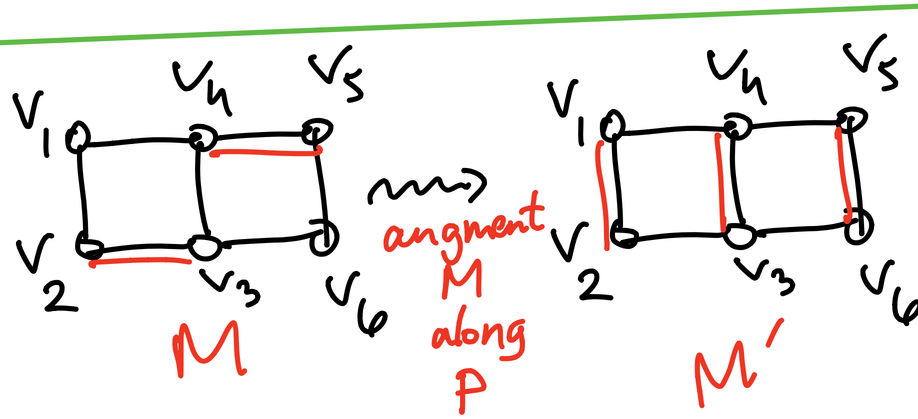


The black edges are in E.
The red edges are in M.
v_1, v_m are M-unmatched

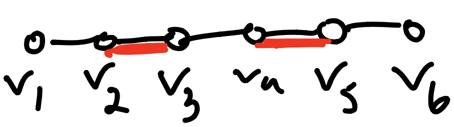
If such an M-augmenting path P exists, create a matching M' having $\#M' = 1 + \#M$ by replacing P with



e.g.



$P =$

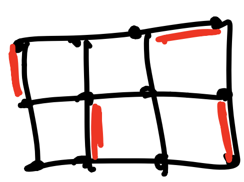


THEOREM: M is a maximum (-sized) matching in G $\iff$ $\nexists$ any M-augmenting path P in G.

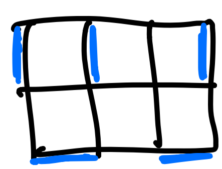
proof: ($\Rightarrow$) follows from what we just observed: if such a P exist, augment M to show it was not max-sized.

($\Rightarrow$): If M is not maximum-sized, say $\exists$ M' a matching with $\#M' > \#M$, then we'll convince ourselves an M-augmenting path P exists, by showing P is one of the connected components of the (mult-)graph $(V, M \sqcup M')$.

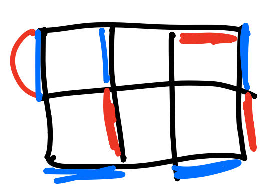
e.g.

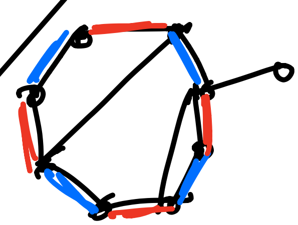


M

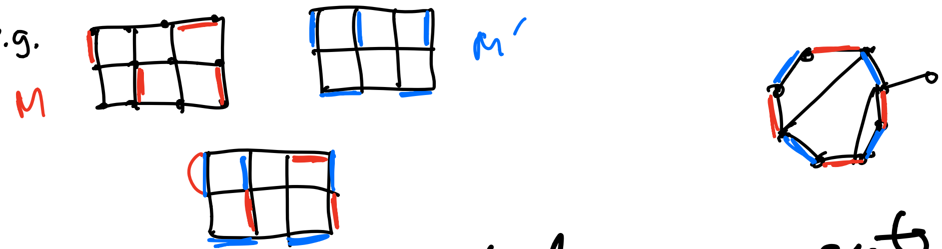


M'



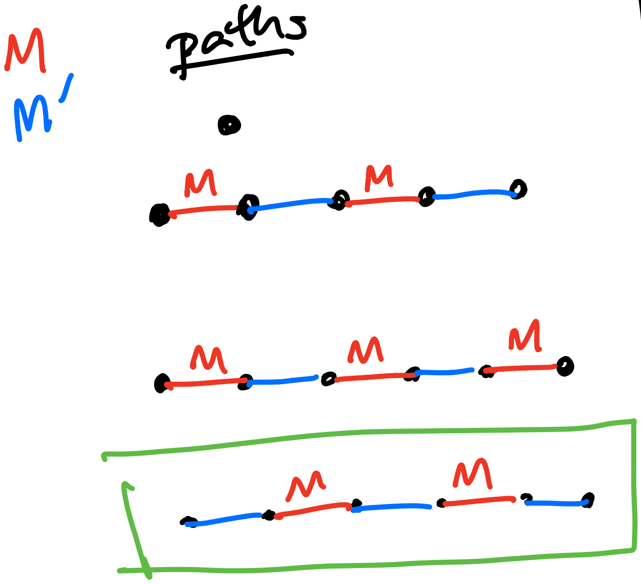


e.g.

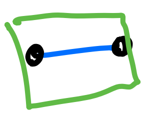


What <u>can</u> the connected components of $(V, M \mathbin{\dot\cup} M')$ look like?

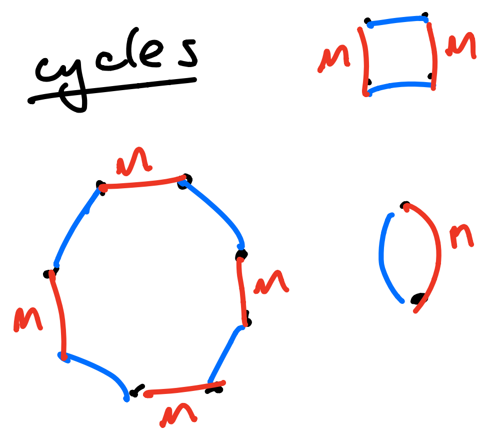
<u>paths</u>



These are all M-augmenting paths!



<u>cycles</u>



(<u>LEMMA</u>: A graph with $\deg(v) \leq 2$, $\in \{0,1,2\}$ has as connected components only paths & cycles)

<u>CLAIM</u>: The green kind of connected component must appear in $(V, M \mathbin{\dot\cup} M')$ because they're the only ones with strictly more M' edges than M edges, and $\#M' > \#M$! ∎

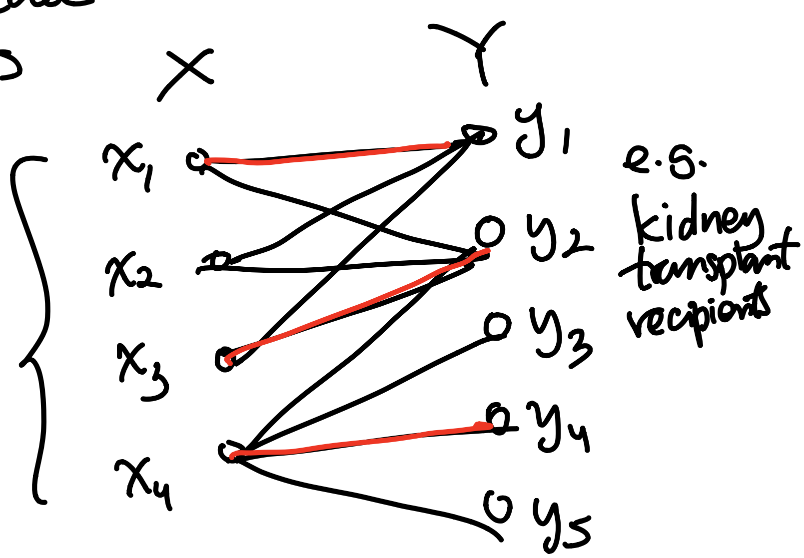
THEOREM: M is a maximum (-sized) matching in G $\iff$ $\nexists$ any M-augmenting path P in G.

The max matching algorithms start with any matching M in G and search for M-augmenting paths. If they find one, they augment along it. If they find none and can prove it, they stop with M provably max-sized.

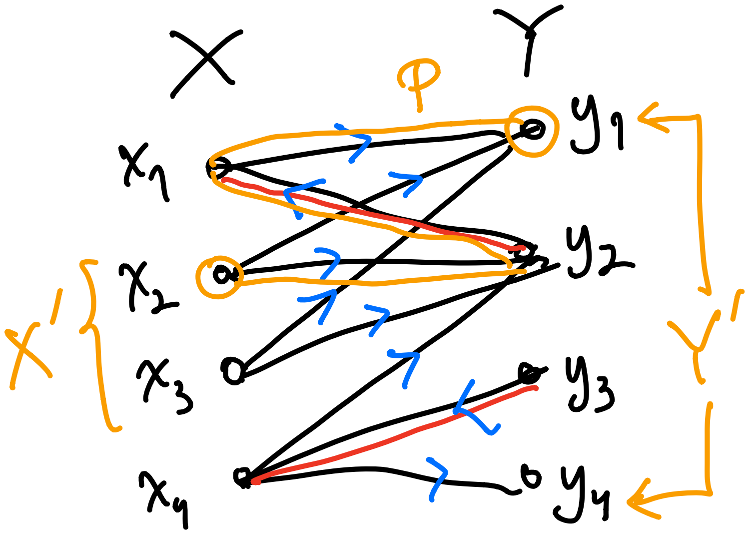
The algorithm is easier (and faster) if M is bipartite, but still fast in general (called Edmonds's Blossom Algorithm in general).

The bipartite case is most important in applications

$V = X \sqcup Y$

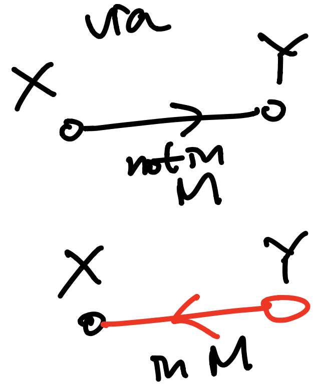


How to find an M-augmenting path in a bipartite graph G



M

Make a digraph from G and M via



Now look for any directed path P from a vertex in $X' :=$ M-unmatched vertices of X to $Y' :=$ M-unmatched vertices of Y

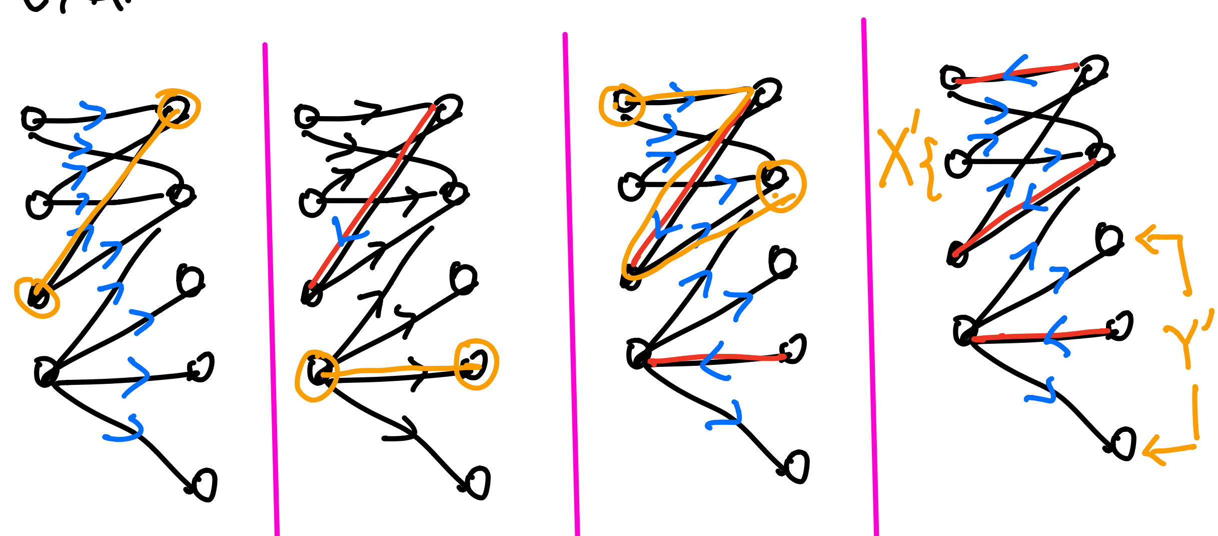
CLAIM:
This is the <u>same</u> (forgetting the arrows) as an M-augmenting path P.

This then leads to the Hungarian algorithm for finding max matchings in bipartite graphs

Start with $M_0 = \emptyset$ (no edges)

Given M_i with i edges, build the digraph, check for M_i-augmenting paths, and either augment if they exist, or stop with M_i max-sized.

EXAMPLE



G

$M_0 = \emptyset$ $\quad$ M_1 $\quad$ M_2 $\quad$ STOP M_3

Can't get X' to Y' following arrows, so M_3 is max-size.

A non-trivial corollary...

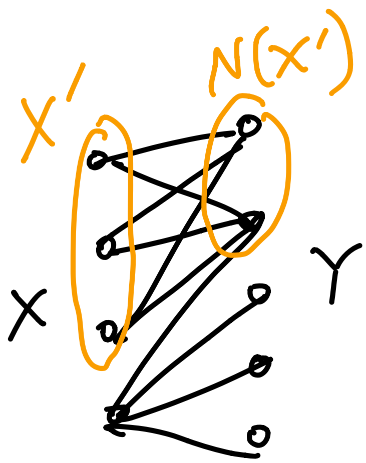
<u>THEOREM</u> (Hall's Marriage Theory)

In a bipartite graph $G = (\underbrace{X \sqcup Y}_{V}, E)$

there will be a matching M that matches <u>all of</u> X (i.e. $\#M = \#X$)

$\iff \forall$ subsets $X' \subseteq X$

$$\#\underbrace{N(X')}_{\substack{\text{neighbors of } X' \\ := \text{neighbors} \\ \text{of at least one} \\ x' \in X'}} \geq \#X'$$



<u>proof</u>: next time,

using Hungarian algorithm...