1. Overview

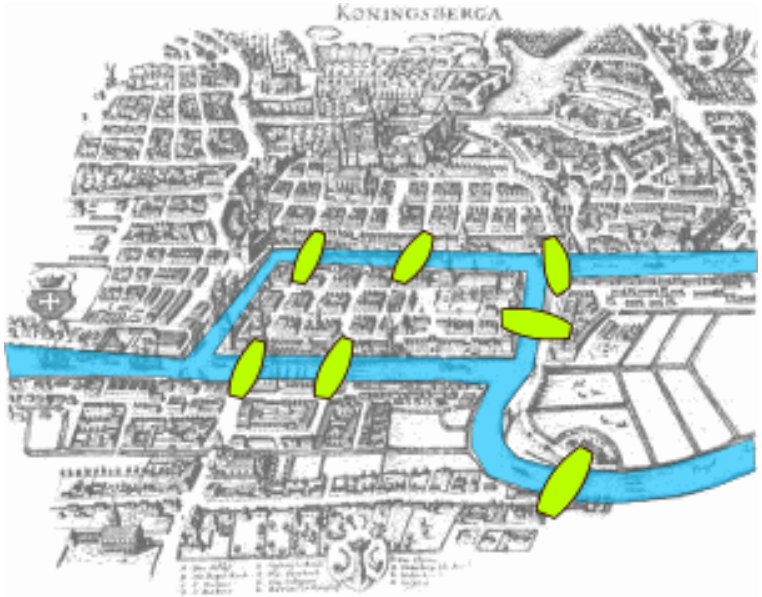
- Eulerian cycles
- Minimum spanning trees

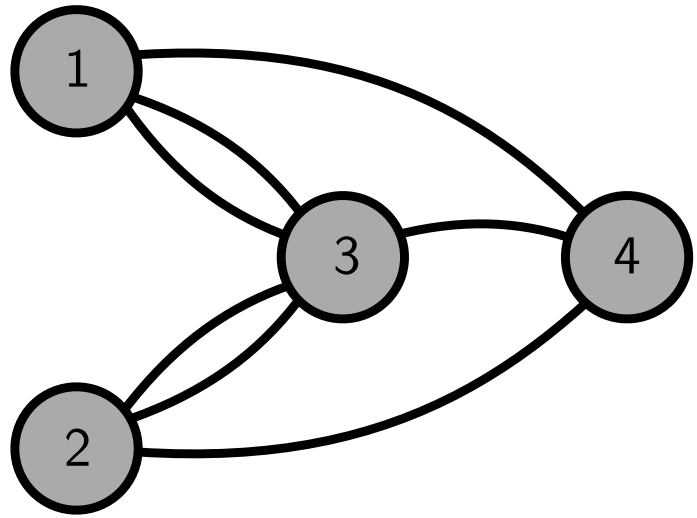
Eulerian cycle

Graph theory

Seven bridges of Königsberg, [Leonhard Euler 1735]

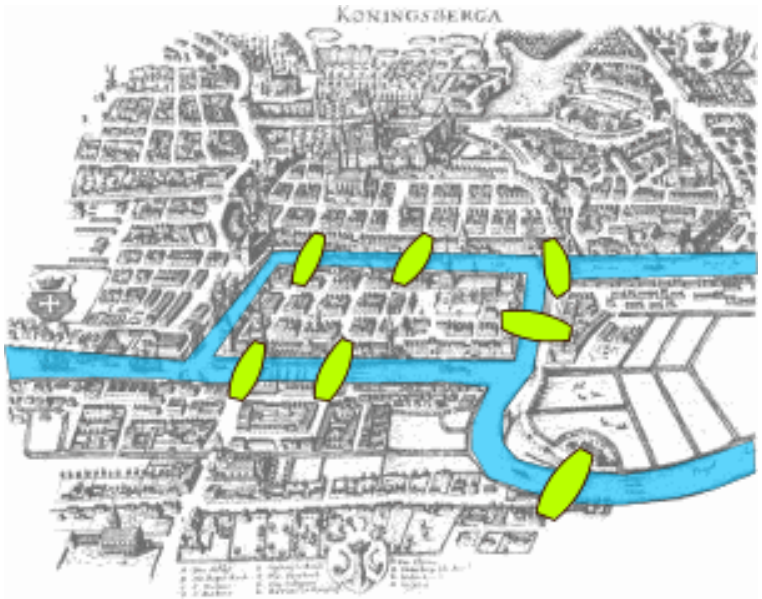
Can you cross all seven bridges exactly once?

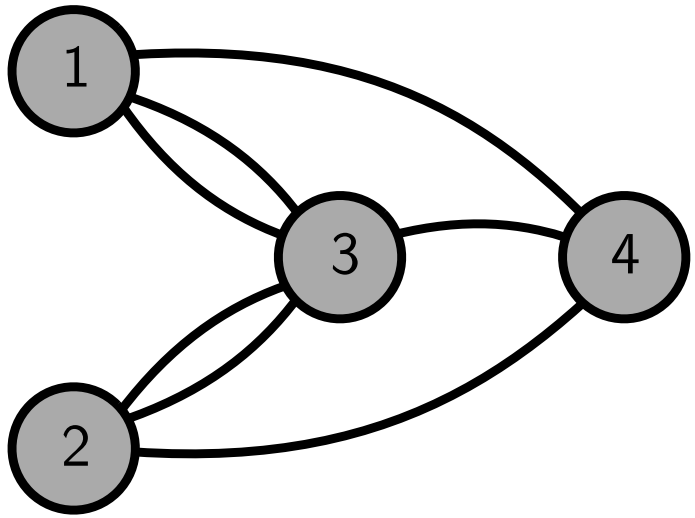




Graph $G = (V, E)$

- ★ V: set of vertices $\{1, \ldots, n = |V|\}$
- ★ E: set of edges $\{(i, j), \ldots\}$ or $\{e_1, e_2, \ldots, e_{m=|E|}\}$
- ★ *directed edge* is an ordered pair such that $(i, j) \neq (j, i)$
- ★ *undirected edge* is an unordered pair such that $(i, j) = (j, i)$
- ★ *multigraph* is a graph with multiple edges between a pair of nodes
- ★ *loop* or a self-loop is an edge between a node and itself
- ★ *simple* graph is a graph with no loops and no multi-edges





Representing and storing a Graph $G = (V, E)$

- Adjacency matrix
- Incidence matrix
- Edge list

Graph terminology

- ★ two nodes are said to be *adjacent* or *neighbors* if they are connected by an edge
- ★ *walk*: $v_1 v_2 \cdots v_k$ such that $(v_i, v_{i+1}) \in E$
- ★ *path*: a walk $v_1 v_2 \cdots v_k$ such that $v_i \neq v_j$
- ★ *closed walk*: a walk such that $v_1 = v_k$
- ★ *cycle* is a closed path
- ★ a graph is *connected* if there exists a path from any node i to any j
- ★ *degree* of a node is the number of adjacent nodes

For the seven bridges of Königsberg example,

- ★ Eulerian walk: a walk that includes all the edges exactly once (but we will also call such a walk a *Eulerian path*)
- ★ Eulerian cycle: an Eulerian walk that is closed (precisely, it should be called Eulerian closed walk)

Q1 given a graph, can we decide whether there is an Eulerian cycle or not?

Q2 if there is one, how can we find one efficiently?

Theorem. there exists an Eulerian cycle if and only if the graph is connected and every node has an even degree

- "$\Rightarrow$" only if part is easy: it follows from "all closed walks have even degrees"
- "$\Leftarrow$" if part: a constructive proof due to Fleury, 1883

1. algorithm:
   - ★ start at any node
   - ★ at each step, choose the next edge in the path to be the one whose deletion would not disconnect the graph
   - ★ if there is no such edge, pick the remaining edge left at current node
   - ★ move to the chosen node and delete the edge
   - ★ if there is no edge left in the end, the sequence of edges traversed is an Eulerian cycle
   - ★ if there are edges left that cannot be traversed, then the graph has no Eulerian cycle

fact 0. sum of all the degrees is even, i.e. $\sum_{i \in V} d_i = 2|E|$

2. correctness:

fact 1. if the original graph has all even degrees, then all node degrees remain even, except for the start node and current node

fact 2. hence, there is always an edge to move to, and the process only fails to find an Eulerian cycle only if at a certain step all edges of the current node result in disconnected graph

fact 3. this only happens when current node has degree one, in which case disconnected part is a single isolated node.

proof of fact 3. We prove by contradiction. Suppose current node has degree d larger than 2, and removing any of these edges result in a disconnected graph. Then, the graph looks like a star where removing the current node results in d disconnected subgraphs. Let $G_1 = (V_1, E_1), \ldots, G_d = (V_d, E_d)$ denote these subgraphs, with the current node removed. We know that there are even number of odd degree nodes in each of these G_i's. It follows that if we put the current node back in, then there are odd number of odd degree nodes in each V_i's. In particular, there is at least one odd degree node in each subset of nodes $V_1, \ldots, V_d$. Hence, in the graph G (which is the graph with remaining edges at current step of the algorithm), there are at least $d+1$ odd degree nodes counting the current node. This is a contradiction if $d > 2$, since we know that there are only two odd degree nodes from Fact 1.

3. complexity (running time): $|E|$ steps, but even with the best known algorithm for bridge-finding gives $T(G) = O(|E| \log^3 |E| \log\log |E|)$

   a more efficient algorithm: Hierholzer, 1873

   - ★ idea: augmenting closed walks
   - ★ complexity: $O(|E|)$

   Corollary. An undirected graph has an *Eulerian path* iff it is connected and only two nodes have odd degrees.

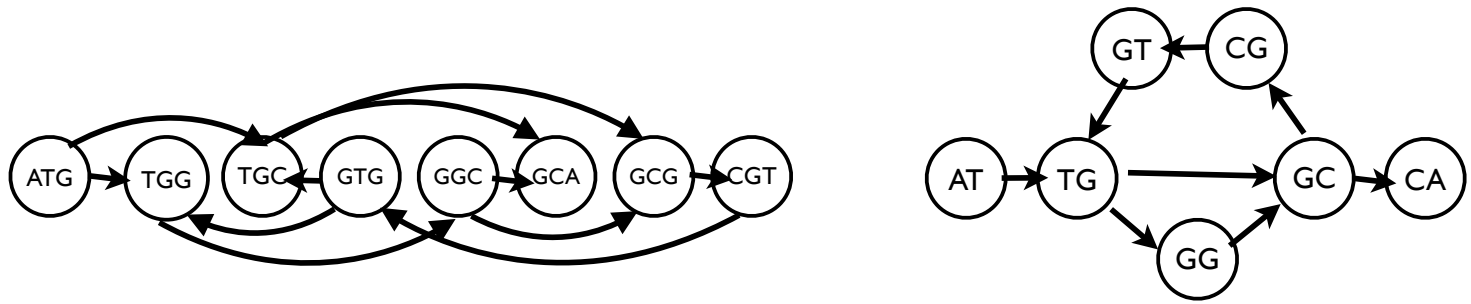
   Theorem. A *directed* graph has an Eulerian cycle off it is connected and the in-degree is equal to the out-degree for all nodes.

   Similarly, a *Hamiltonian path* is a path that passes each node exactly once
   However, the best known algorithm for finding a hamiltonian path has exponential complexity $\Omega(e^n)$ (NP-complete)

Application: DNA sequencing

DNA is a chain of complementary nucleotides. There are 4 different nucleotides: A, G, C, and T. One technique to sequence DNA, introduced by Professor Patrick Brown, is to use a collection of small subsequences of length k. DNA sequence is cut into pieces, and tagged with a fluorescent agent, then exposed to an micro-array with known subsequences. One can detect the presence of particular subsequences.

Given a DNA sequence s and all detected set of length k subsequences $\sigma = \{\sigma_1, \ldots, \sigma_{n-k+1}\}$, we want to reconstruct s from σ.
For example, $\{ATG, TGG, TGC, GTG, GGC, GCA, GCG, CGT\}$



- ★ s is a Hamiltonian path (left): node is σ_i, edge if $k-1$ overlap
- ★ s is a Eulerian path (right): node is a length $k-1$ subsequence, edge if σ_i

Pseudo code for Eulerian cycle algorithm

- Input: $G(V_0, E_0)$
- Output: Eulerian cycle $w = (w_1, \ldots, w_{m+1})$

- pick $v \in V$
- set $E \leftarrow E_0$
- $w \leftarrow ()$
- repeat until no more edge to move
  - for each $(v, u) \in E$
    - ⋆ if removing (v, u) does not disconnect $G(V_0, E)$ or if there is only one edge (v, u) in E then
    - ⋆ remove (v, u) from E
    - ⋆ $w = w + (v, u)$
    - ⋆ $v \leftarrow u$
    - ⋆ else break
- end repeat
- if E is empty output w
- else declare no Eulerian cycle

Graphs, Networks, and Algorithms

overview of the course

- Part 1: network algorithms
  - ★ learn many interesting problem on graphs and networks (motivated by real applications)
  - ★ identify which problems are solvable and which are not
  - ★ algorithms for exactly or approximately solving the problems
  - ★ analyze those algorithms: correctness and complexity
- Part 2: neural network on graphs
  - ★ apply neural network to solve the problems we learned in part 1

examples

- minimum spanning tree
- matching
- maximum flow
- spectral methods
- linear programming

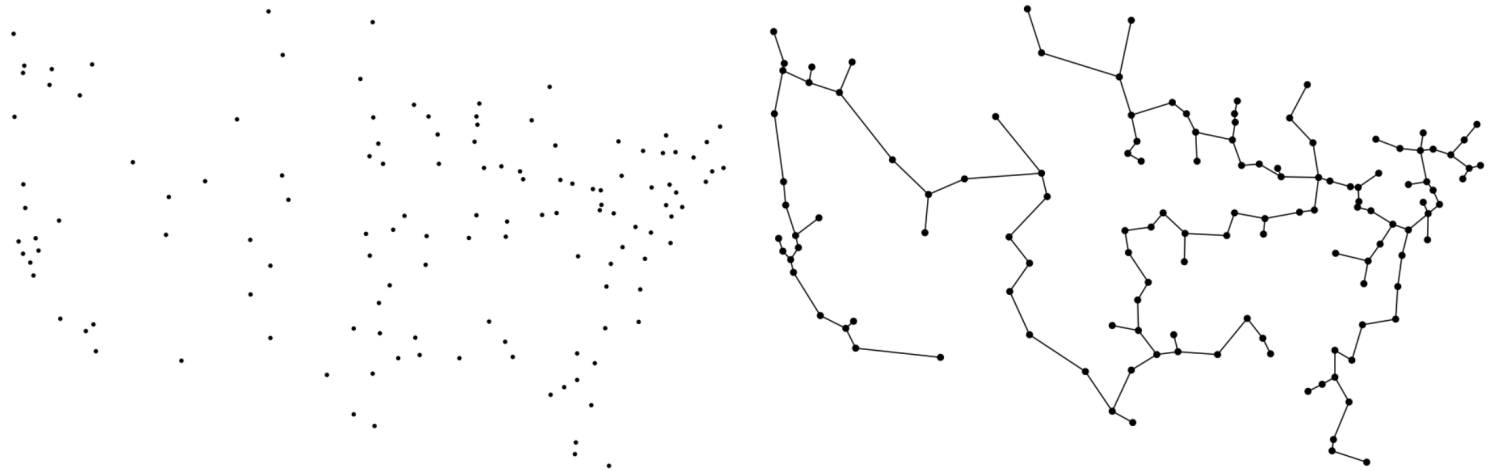
Minimum spanning trees

- tree: a connected graph with no cycle
- leaf: a node in a tree with degree one
- **claim 1** the number of edges in a tree of size n is $n-1$ (Homework 0)
- **claim 2** every finite size tree of size at least two has at least two leaves
- **proposition.** if a graph has two of the following three properties, it has all three
  1. G is connected
  2. G has no cycle
  3. $|E| = |V| - 1$

  Therefore, any graph with any two of the above properties is a tree

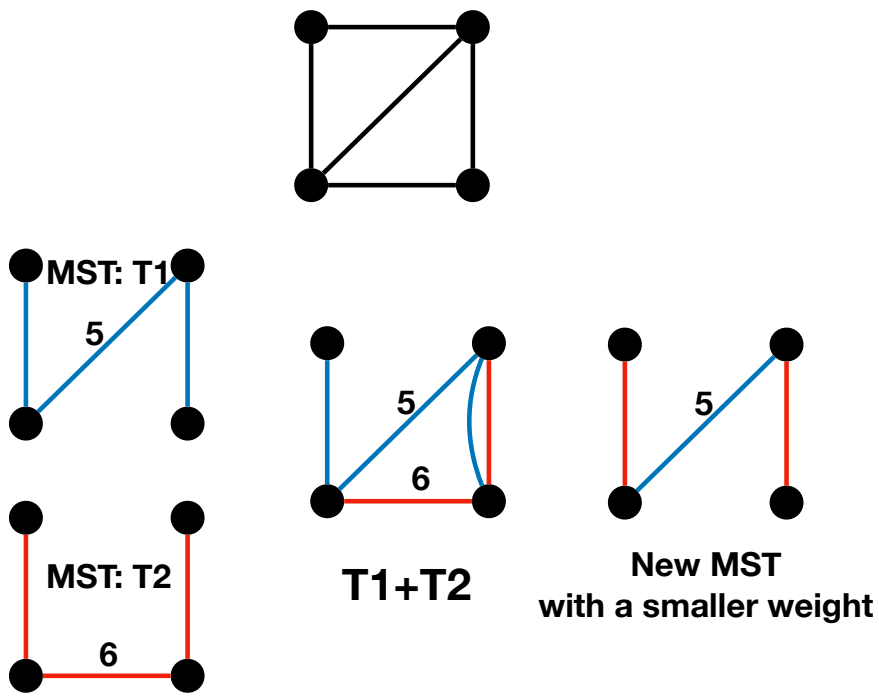
  proof.
  - $(1), (3) \Rightarrow (2)$: we prove by contradiction. Suppose there are cycles. Then, we can remove an edge in a cycle and the resulting graph is still connected. we remove edges until we get a connected graph with no cycle. Then by claim 1., there must be $n-1$ edges. However, this is a contradiction. We removed (some positive number of) edges starting from $n-1$ edges. So it is impossible that we end up with $n-1$ edges.
  - $(2), (3) \Rightarrow (1)$: Similarly, assume (2), (3) and suppose (1) is false.

Minimum spanning trees



- **definition. spanning tree** of a connected undirected graph: a tree composed of all vertices and a subset of the edges
- **definition. minimum spanning tree** of a weighted connected undirected graph: a spanning tree with minimum weight
- motivation: design of an efficient connected network (e.g. electrical grid, transportation network), phylogeny

- **definition. Cut**: a cut (S, S') is a partition of V such that $S \cap S' = \emptyset$ and $S \cup S' = V$. The set of edges between S and S' is also called a cut or a cut-set.
- **definition. Forest**: a forest is a collection of disconnected trees
- properties of a minimum spanning tree
  - in general, MST is not unique
  - **Uniqueness**: if each edge has a distinct weight, then MST is unique.
    - ★ Proof by contradiction:

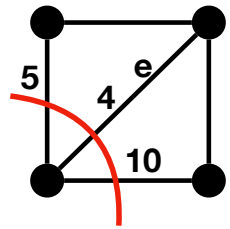


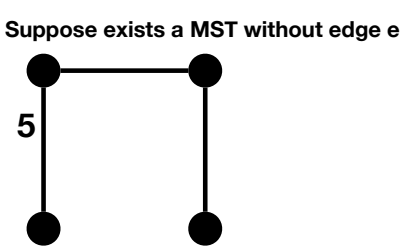
- **Uniqueness**: if each edge has a distinct weight, then MST is unique.
  - Proof by contradiction:

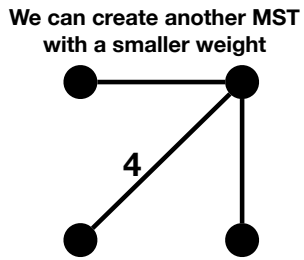
    Assume there are two MSTs T_1 and T_2. Consider one edge e_1 which is included T_1 but not in T_2. Also consider another edge e_2 which is in T_2 but not in T_1, and makes a cycle when added to T_1 such that the cycle includes e_1.

    Without loss of generality, let e_1 be the one with *strictly* smaller weight than e_2. Then, by removing e_2 from T_2 and adding e_1, we can create a new spanning tree that has strictly smaller weight than T_2. This contradicts with our assumption that T_2 is a minimum spanning tree.

- **Cut property**: for any cut C in the graph, if the weight of an edge e of C is smaller than any other edges of C, then e belongs to all MST's.
  - ★ Proof by contradiction:







Assume that there exists a cut such that the minimum weight edge (i, j) in that cut is not in a MST. Then, there is another edge (k, ℓ) in the cut which is included in the MST. If we create another spanning tree from the MST by eliminating (k, ℓ) and adding (i, j), then the weight of this new spanning tree is smaller than the original MST. This violates the assumption that the original tree was a MST.

- **Minimum-cost edge**: If an edge e is the edge with unique minimum cost in a graph, then e is included in all MST.
  - ★ This follows from the Cut property, since there is a cut that includes this min-cost edge.

How can we find one of MSTs in a weighted undirected graph?

Algorithms for finding a MST

- Prim's algorithm, 1957
  1. Initialize $E_T = \{\}$ and $V_T = \{i\}$ with any single node i
  2. Grow the tree by one edge: of all the edges that connect the tree to nodes not yet in the tree, find the minimum-weight edge, and transfer it to the tree.
  3. Repeat until all nodes are in the tree

  correctness of Prim's algorithm follows from the cut property
  complexity of Prim's algorithm: $O(|V|^2)$ using adjacency matrix and distance array

- Kruskal's algorithm, 1956
  1. Initialize $E_T = \{\}$ and $V_T = V$
  2. Sort the edges such that $c(e_{i_1}) \leq \cdots \leq c(e_{i_m})$
  3. While $|E_T| < |V| - 1$, add the cheapest unused edge that does not create a cycle and discard the chosen edge and those edges that create cycles from the candidate set
- correctness of Kruskal's algorithm follows from the cut property, since we are adding the minimum weight edge in a cut
- complexity of Kruskal's algorithm: $O(|E| \log |E|) = O(|E| \log |V|)$ if we use the best union-find data structure

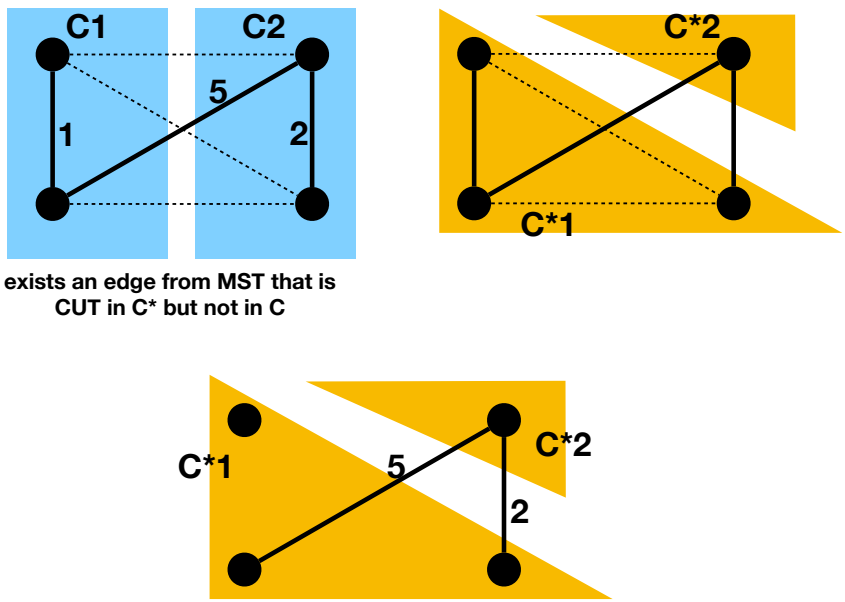
- Application: clustering in bio-informatics

  DNA arrays measure gene expressions. One can use these gene expressions to compute a distance $d(i,j)$ between a pair of genes g_i and g_j. This distance $d(i,j)$ records how close, or similar, genes g_i and g_j are, based on their expression levels. Given n nodes (=genes) and distance between all pairs of nodes, clustering problem aims to partition the nodes into k groups, such that the nodes in the same group are closer compared with nodes in different groups. Consider a clustering $C = \{C_1, \ldots, C_k\}$, where C partitions the nodes into k groups. Then we can formulate the clustering problem as

  $$\text{maximize}_C \;\; D(C)\,,$$

  where $D(C) \triangleq \min_{i,j} D(C_i, C_j)$ and $D(C_i, C_j) \triangleq \min_{u \in C_i, v \in C_j} d(u,v)$.

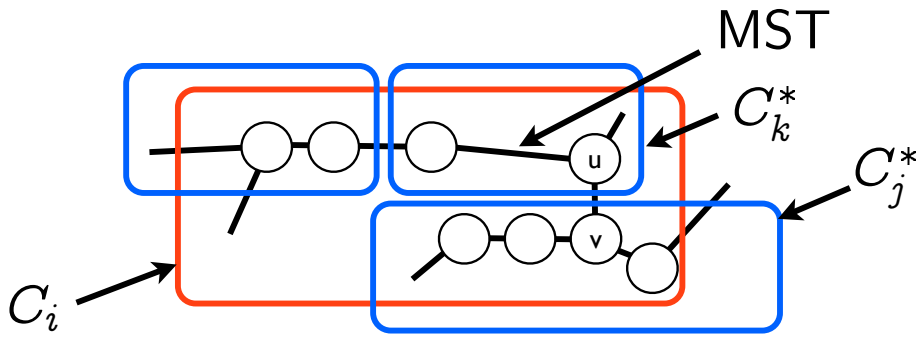
- Consider the following MST based clustering. Given a minimum spanning tree, delete the most expensive $k-1$ edges in the MST. Then, let C denote the k partition resulting from the k connected components of the deletion.
  Theorem. The partition C of the MST based clustering is the optimal solution to maximizing $D(C)$.
  proof.
  We prove by contradiction. Assume there is another clustering $C^* = \{C_1^*, C_2^*, \ldots, C_k^*\}$ such that it achieves larger utility: $D(C^*) > D(C)$.



Theorem. The partition C of the MST based clustering is the optimal solution to maximizing $D(C)$.

proof.

We prove by contradiction. Assume there is another clustering $C^* = \{C_1^*, C_2^*, \ldots, C_k^*\}$ such that it achieves larger utility: $D(C^*) > D(C)$.



One fact is that by the cut property of a MST, for a pair of cluster C_i and C_j, the deleted edge of MST that was connecting these two sets has the minimum weight among all edges between C_i and C_j. Consider two nodes u and v, which are adjacent in the MST and in the same cluster C but different clusters in C^*. Since we deleted largest edges in MST to get C, we know

$$d(u, v) \leq \min_{a,b} D(C_a, C_b) = D(C)$$

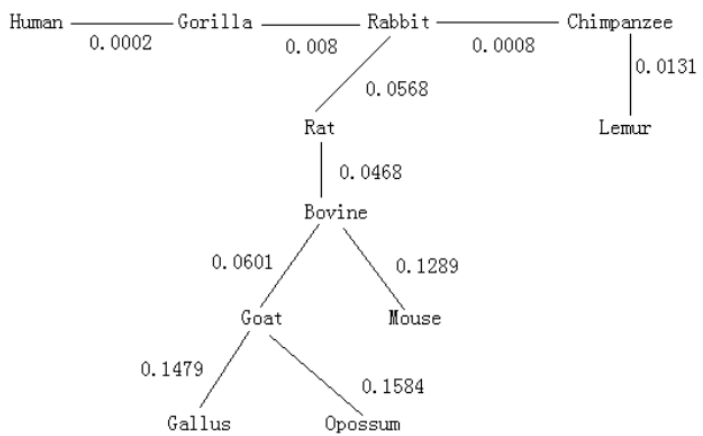
Also, by definition, $D(C^*) \leq D(C_k^*, C_j^*) \leq d(u, v)$. This contradicts the supposition that $D(C^*) > D(C)$.

Application: MST gives a heuristic for phylogeny (tree of life)

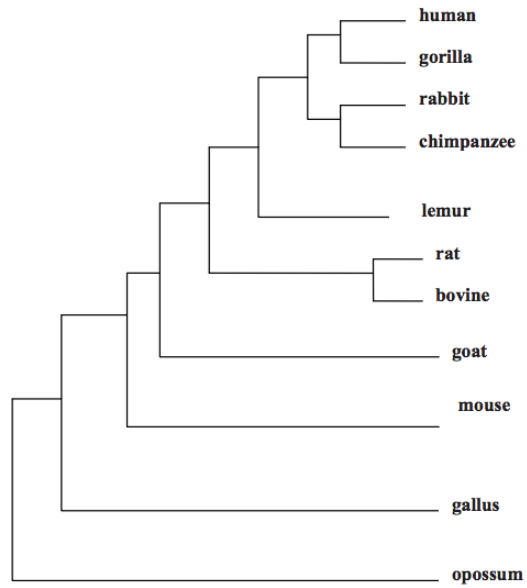
[A method for constructing phylogenic tree based on MST, Yang et al. 2011]

Species	Human	Goat	Gallus	Opossum	Lemur	Mouse	Rabbit	Rat	Bovine	Gorilla	Chimpanzee
Human	0	0.1254	0.4844	0.3571	0.0425	0.0594	0.0108	0.0292	0.0589	0.0002	0.0117
Goat		0	0.1479	0.1584	0.0387	0.2746	0.0789	0.1754	0.0601	0.1172	0.0830
Gallus			0	0.1601	0.2749	0.7214	0.3785	0.5838	0.3635	0.4681	0.3797
Opossum				0	0.2023	0.5831	0.2969	0.3914	0.2528	0.3450	0.2865
Lemur					0	0.1815	0.0137	0.0899	0.0537	0.0368	0.0131
Mouse						0	0.1103	0.0559	0.1289	0.0663	0.1167
Rabbit							0	0.0568	0.0575	0.0080	0.0008
Rat								0	0.0468	0.0316	0.0604
Bovine									0	0.0570	0.0641
Gorilla										0	0.0087
Chimpanzee											0

distance matrix between genes



corresponding MST



Phylogenic tree

Example: minimax path problem

[Network Flows, Ahuja, Magnanti, Orlin, page 513]

On a weighted undirected graph $G = (V, E, \{w_{ij}\})$, define the value of a path $P = p_1 p_2 \cdots p_k$ from node p_1 to node p_k as the maximum weight of an edge in P:

$$V(P) \quad = \quad \max_{i=1,\ldots,k-1} \; w_{p_i, p_{i+1}}$$

The **minimax path problem** is to find, for every pair of nodes i and j, a minimum value path from node i to j. Let T be the minimum spanning tree on G.

Theorem. The unique path between i and j in the MST T is the minimum value path between i and j.

proof. Let us focus on a particular pair of nodes i and j. Let P be the unique path between i and j in T. Let (k, ℓ) be the maximum weight edge in P. Then, removing (k, ℓ) from T creates two partitions S and S', such that $i \in S$ and $j \in S'$. This defines a cut (S, S'). For any edge in the cut (i', j') such that $i' \in S$ and $j' \in S'$, the cut property of a MST implies that

$$w_{k,\ell} \leq w_{i',j'}$$

Since any path P' between i and j must contain one of the nodes from the cut (S, S'), $w_{k,\ell}$ is the value of path P and P is the minimum value path. Hence, this establishes that the unique path in MST T is the minimum value path between all pairs of nodes.

Examples 1

- Problem 1. [Network Flows, Ahuja, Magnanti, Orlin, page 513]
  An intelligence service has n agents in a non friendly country. Each agent knows some of the other agents and can exchange messages with them. For each message exchanged between agents i and j, the message will fall into hostile hands with a certain probability p_{ij} that is known to us. The leader wants to pass a message to everyone while maximizing the total probability that the message is not intercepted, where the probability of the message not being intercepted is

$$\mathbb{P}(\text{message not intercepted}) \quad = \quad \prod_{(i,j)\in\mathcal{E}} (1 - p_{ij}) \,,$$

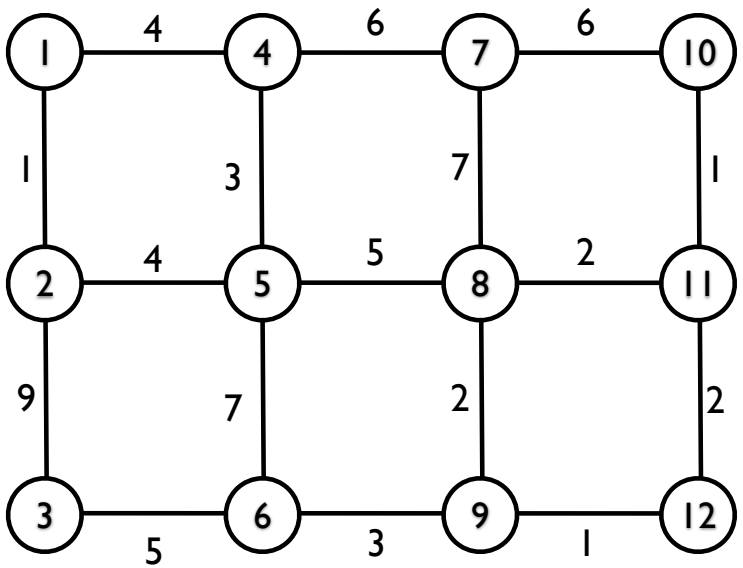
  where $\mathcal{E}$ is the set of all pairs of agents exchanging messages.

  Consider an undirected graph, where each node is an agent and an edge indicates that those two agents can exchange messages. That is two agents who are not connected by an edge cannot exchange messages.

  Explain how to find the set $\mathcal{E}$ of pairs of agents that needs to exchange messages, such that the probability of message being intercepted is minimized and every agent has the messages when no interception occurs. Specifically, formulate this problem as a maximum spanning tree problem, and explain your answer. (note that maximum spanning tree algorithm maximizes the sum of the weights and not the product of the weights in the spanning tree.)

Example 1

- Problem 2. [Network Flows, Ahuja, Magnanti, Orlin, Ex.13.6] Consider the following network of a highway map, and the number on the edge is the maximum elevation encountered in traversing the edge. A traveler plans to drive from node 1 to node 12 on this highway. This traveler dislikes high altitudes and so would like to find a path connecting node 1 to node 12 that minimizes the maximum altitude. Formulate this as a minimum spanning tree problem and find the best path for this traveler.



Examples 1

Problem 3.

We proved in class the **cut property** of a Minimum Spanning Tree (MST). Let cut C be the collection of edges between two partition of vertices (S, S^c). Then, if an edge in C has smaller weight than any other edges in C, it belongs to all MSTs of this graph.

(a) In this problem we prove the **cycle property** of a MST. Show that in any cycle C in the graph, if an edge has larger weight than any of the other edges in C, then this edge cannot belong to an MST.

(b) In this problem, we prove a sufficient condition for uniqueness of a MST. First, show that a graph has a unique minimum spanning tree if, for every cut of the graph, the edge with the smallest weight across the cut is unique. Next, show that the converse is not true by giving a counter-example.

(c) In this problem, we prove a sufficient condition for uniqueness of a Minimum Spanning Tree. First, show that a graph has a unique minimum spanning tree if, for every cycle in the graph, the edge with the largest weight in the cycle is unique. Next, show that the converse is not true by giving a counter-example.

Examples 1

Problem 4.

Prove that at least one of G or $\overline{G}$ is connected. Here, $\overline{G}$ is a graph on the vertices of G such that two vertices are adjacent in $\overline{G}$ if and only if they are not adjacent in G.

Problem 5.

Given a graph G, two players play the following game. Destroyer plays first and and removes an edge of his choice. Then Connector takes his turn and fixes an edge of his choice. A removed edge cannot be fixed anymore and a fixed edge cannot be removed anymore. The players alternate, one edge at a time. Connector wins if he fixes a spanning tree in the graph. Destroyer wins if he manages to disconnect the graph. (It's easy to see that exactly one player wins at some point.) Prove that Connector has a winning strategy, if and only if G contains two edge-disjoint spanning trees.