Lecture 06
Spanning Trees and Enumeration

Dr. Rajiv Misra
Associate Professor
Dept. of Computer Science & Engg.
Indian Institute of Technology Patna
rajivm@iitp.ac.in

Preface

Recap of previous Lecture:

- In the previous lecture, we have discussed the basic properties of trees and distance.

Content of this Lecture:

- In this lecture, we will discuss the Prüfer code Cayley's formula, counting of spanning trees using various methods including Matrix Tree Theorem.

Spanning Trees and Enumeration 2.2

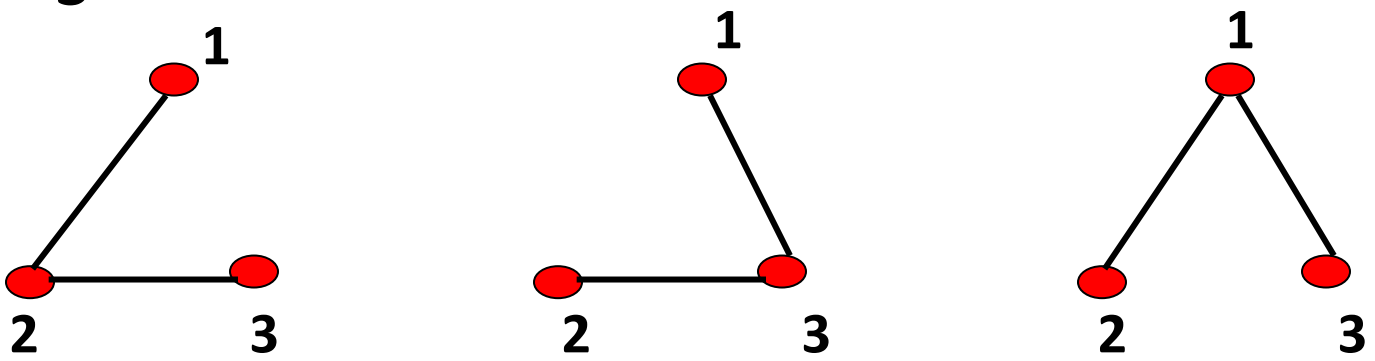
- There are $2^{\binom{n}{2}}$ simple graphs with vertex set $[n]=\{1,\ldots,n\}$, since each pair may or may not form an edge.

- How many of these are trees?

- With vertex set *[n]*, there are n^{n-2} labelled trees; this is **Cayley's Formula**.

Enumeration of Trees 2.2

Example: $n = 3$



- With one or two vertices, only one tree can be formed.
- With three vertices there is till only one isomorphism class, but there are three trees with vertex set [3].
- With vertex set [4], there are four stars and 12 paths, yielding 16 trees.
- With vertex set [5], a careful study yields 125 trees.

Cayley's Formula and Prüfer Code

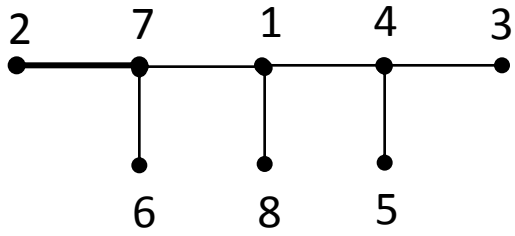
- Now we may see a pattern. With vertex set *[n]*, there are n^{n-2} trees; this is **Cayley's Formula**. Prüfer, Kirchhoff, Pólya, Renyi, and others found proofs.
- We will discuss a bijective proof, establishing a one-to-one correspondence between the **set of trees with vertex set *[n]*** and **a set of known size.**
- Given a set *S* on *n* numbers, there are exactly n^{n-2} ways to form a list of length *n-2* with entries in *S*. The set of lists is denoted S^{n-2} .
- We use S^{n-2} to encode the trees with vertex set *S*. The list that results from a tree is its **Prüfer code.**

Algorithm 2.2.1. (Prüfer code) Production of $f(T) = (a_1,\ldots,a_{n-2})$

Algorithm. (Prüfer code) Production of $f(T) = (a_1,\ldots,a_{n-2})$

Input: A tree T with vertex set $S \subseteq N$

Iteration: At the ith step, delete the least remaining leaf, and let a_i be the ***neighbor*** of this leaf.



1. Delete 2 $a_1 = 7$
2. Delete 3 $a_2 = 4$
3. Delete 5 $a_3 = 4$
4. Delete 4 $a_4 = 1$
5. Delete 6 $a_5 = 7$
6. Delete 7 $a_6 = 1$

Generation of Prüfer Sequence from Labelled Tree

Proof Idea: Labelled tree $T \leftrightarrow$ Prüfer sequence S

1:1 correspondence — on n elements of length n-2

- How many such sequence ? $n \times n \times n \times \ldots\ldots \times n = n^{n-2}$

Algorithm: Labelled tree T → Prüfer Sequence S

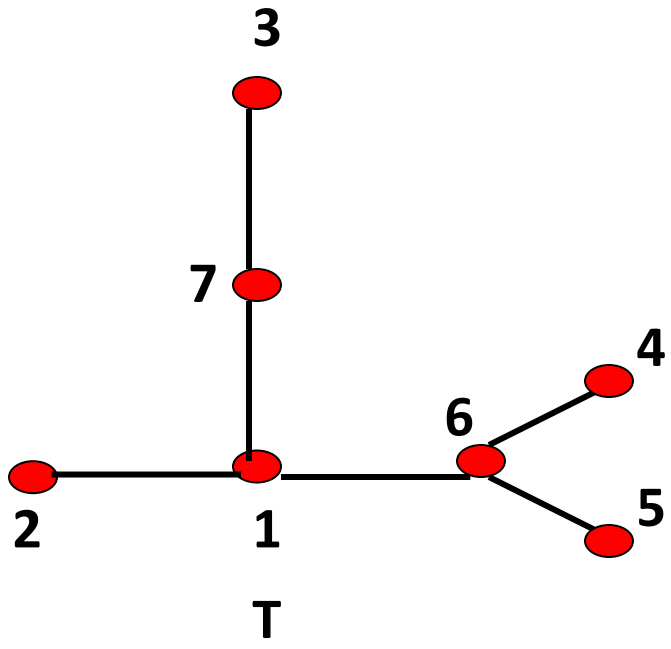
- Find a leaf of T with smallest label

  → add the neighbour to sequence S

  → delete this leaf

- Repeat until our tree is K_2

Example



S = ( )

S = (1)

S = (1, 7)

S = (1, 7, 6)

S = (1, 7, 6, 6)

S = (1, 7, 6, 6, 1)

Why n-2?

Contd…

OBSERVATIONS

- No leaf gets appended to *S*
- Every vertex *v* is added to *S* a total of *deg(v)-1* times

So in given Tree let n vertices, *m* edges ⇨ $m = n - 1$

Number of terms in *S*

$$\sum_{v \in v(T)} (\deg(v) - 1) = \left(\sum_{v \in v(T)} \deg(v) \right) - \sum_{v \in v(T)} (1)$$

$$= 2m - n$$

$$= 2(n-1) - n$$

$$= n - 2$$

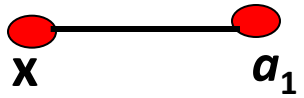
Generation of Labelled Tree from Prüfer Sequence

Proof Idea: Prüfer sequence S of length $n-2$ using symbols $1,\ldots.n$ → Labelled tree T

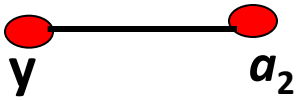
- Let S = $(a_1, a_2\ldots,a_{n-2})$, $a_i \in \{1,\ldots.n\}$

Algorithm: Prüfer Sequence S → Labelled tree T

- Find smallest element $x \in \{1,\ldots.n\}$ $x \notin S$
  - → join x to 1st element of S
  - → delete a_1 from S, delete x from $\{1,\ldots.n\}$



- Find smallest y $\in L \setminus \{x\}$ not in $S \setminus \{a_1\}$
  - → *join y to the* 1st element of $S \setminus \{a_1\}$
  - → delete a_2 from S
  - → delete y from L



- Continue until 2 items remain in L
  - → *join these two with an edge*

Example

S = (1, 7, 6, 6, 1) length 5 → n = 7

L = {1, 2, ……, 7}- vertices with labels

S → T

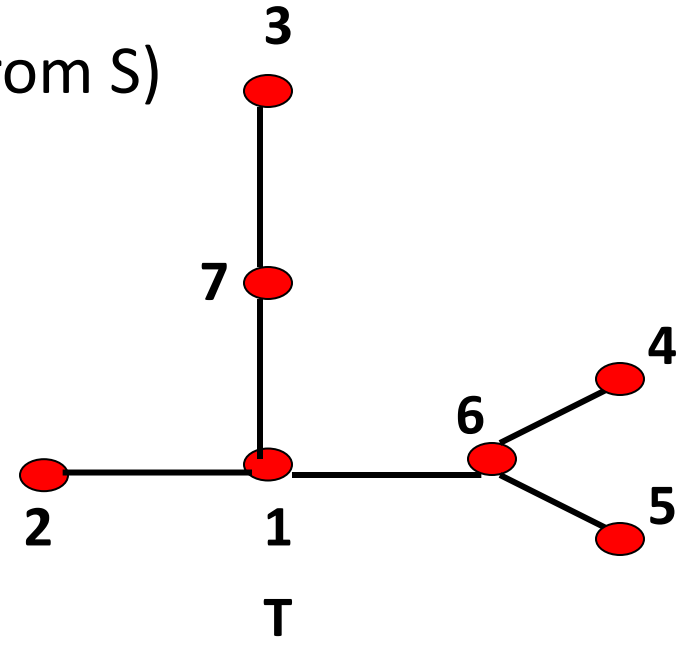
T → S

$x \in L \setminus \{S\}$ *smallest* → x = 2

(edge 12, remove 2 from L and 1 from S)

S = (7, 6, 6, 1)

L = (1, 3, 4, 5, 6, 7}



Theorem: Cayley's Formula [1889] 2.2.3

Theorem: For a set $S \subseteq N$ of size n, there are n^{n-2} trees with vertex set S

Proof: **(Prüfer [1918]).**

This holds for $n = 1$, so we assume $n \geq 2$. We prove that Algorithm 2.2.1 defines a bijection f from the set of trees with vertex set S to the set S^{n-2} of lists of length $n-2$ from S. We must show for each $a = (a_1,\ldots,a_{n-2}) \in S^{n-2}$ that exactly one tree T with vertex set S satisfies $f(T) = a$. We can prove this by induction on n.

Proof by Induction

Basis step: n =2:

- There is one tree with two vertices. The Prüfer code is a list of length 0, and it is the only such list.

Induction step: n > 2:

- Computing *f(T)* reduces each vertex to degree 1 and then possibly deletes it. Thus every nonleaf vertex in *T* appears in *f(T)*.
- No leaf appears, because recording a leaf as a neighbor of a leaf would require reducing the tree to one vertex. Hence the leaves of *T* are the elements of *S* not in *f(T)*. If $f(T) = a$, then the first leaf deleted is the least element of *S* not in *a* (call it *x*), and the neighbor of *x* is a_1.

Contd…

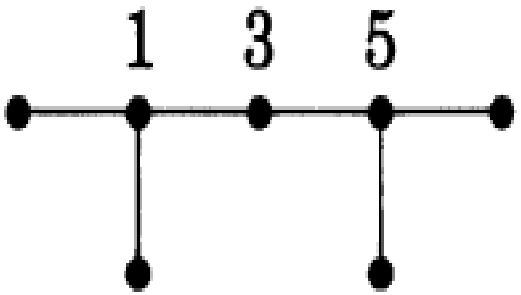
- We are given $a \in S^{n-2}$ and seek all solutions to $f(T)=a$. We have shown that every such tree has x as its least leaf and has the edge xa_1. Deleting x leaves a tree with vertex set $S' = S - \{x\}$. Its Prüfer code is $a' = (a_2,\ldots,a_{n-2})$, an $n-3$-tuple formed from S'.
- By the induction hypothesis, there exists exactly one tree T' having vertex set S' and Prüfer code a'. Since every tree with Prüfer code a is formed by adding the edge xa_1 to such a tree, there is at most one solution to $f(T) = a$. Furthermore, adding xa_1 to T' does create a tree with vertex set S and Prüfer code a, so there is at least one solution.

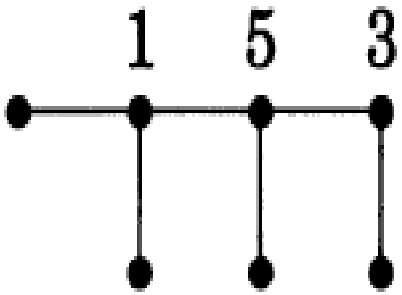
Corollary 2.2.4 count the trees by their vertex degrees.

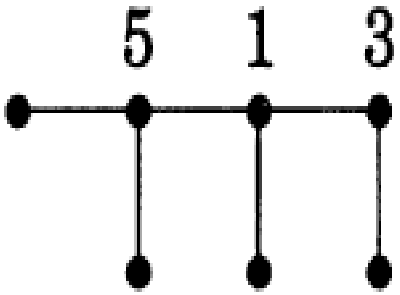
- Given positive integers *d1,….dn* summing to *2n-2*, there are exactly $\frac{(n-2)!}{\prod(di-1)!}$ trees with vertex set *[n]* such that vertex *i* has degree d_i, for each *i*.
- **Proof:** While constructing the Prüfer code of a tree *T*, we record *x* each time we delete a neighbor of *x*, until we delete *x* itself or leave *x* among the last two vertices. Thus each vertex *x* appears $d_T(x) - 1$ times in the Prüfer code.
- Therefore, we count trees with these vertex degrees by counting lists of length *n-2* that for each *i* have $d_i - 1$ copies of *i*. If we assign subscripts to the copies of each *i* to distinguish them, then we are permuting *n-2* distinct objects and there are *(n-2)!* lists. Since the copies of *i* are not distinguishable, we have counted each desired arrangement $\prod(di-1)!$ times, once for each way to order the subscripts on each type of label.

Example

- ***Trees with fixed degrees.*** Consider trees with vertices {1,2,3,4,5,6,7} that have degrees (3,1,2,1,3,1,1), respectively. We compute $\frac{(n-2)!}{\prod(di-1)!}$ = 30; The trees are suggested below
- Only the vertices {1,3,5} are non-leaves. Deleting the leaves yields a subtree on {1,3,5}. There are three such subtrees, determined by which of the three is in the middle.

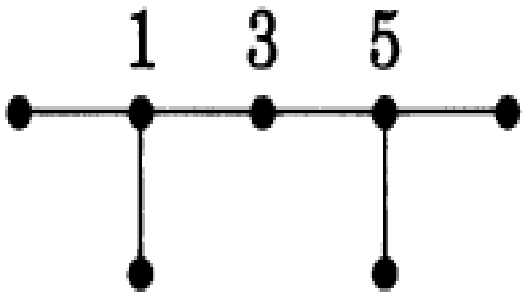


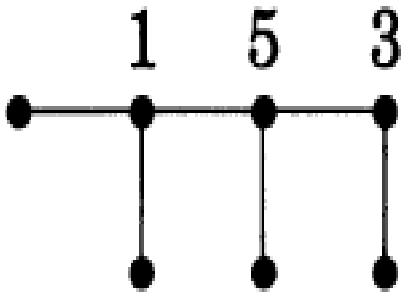


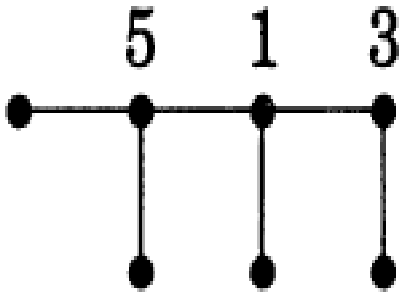


Contd…

- To complete each tree, we add the appropriate number of leaf neighbors for each non-leaf to give it the desired degree, There are six ways to complete the first tree *(pick from the remaining four vertices the two adjacent to vertex 1)* and twelve ways to compute each of the others *(pick the neighbor of vertex 3 from the remaining four, and then pick the neighbor of the central vertex from the remaining three)*







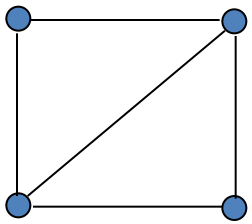
Counting number of Spanning Trees in Graph

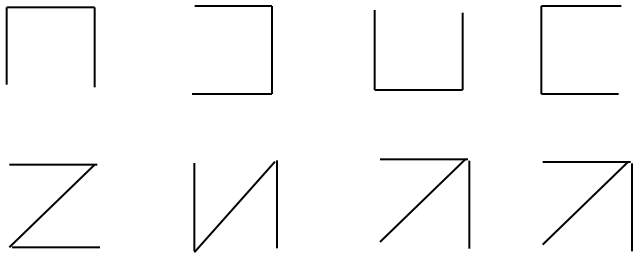
- We can interpret ***Cayley's Formula*** in another way.
- Since the **complete graph** with vertex set *[n]* has all edges that can be used in forming trees with vertex set *[n]*, the number of trees with a specified vertex set of size *n* equals the number of spanning trees in a complete graph on n vertices.
- We now consider the more general problem of computing the number of spanning trees in any graph *G*. In general, *G* will not have as much symmetry as a complete graph, so it is unreasonable to expect as simple a formula as for K_n, but we can hope for an algorithm that provides a simple way to compute the answer when given a graph *G*.

Spanning Trees in Graphs 2.2.6

Example. A kite.

- To count the spanning trees, observe that four are paths around the outside cycle in the drawing
- The remaining spanning trees use the diagonal edge
- Since we must include an edge to each vertex of degree 2, we obtain four more spanning trees.
- The total is eight.



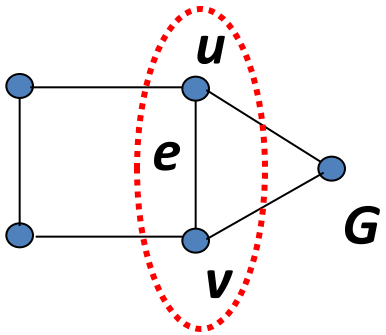


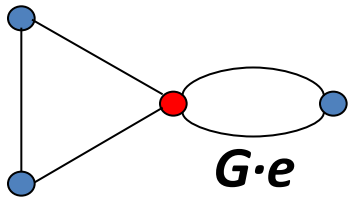
Contd…

- In Example 2.2.6, we counted separately the trees that did or did not contain the diagonal edge.
- This suggests a recursive procedure to count spanning trees.
- It is clear that the spanning trees of G not containing e are simply the spanning trees of G-e, but how do we count the trees that contain e? The answer uses an elementary operation on graphs.

Contraction 2.2.7

- In a graph G, **contraction** of edge e with endpoints u, v is the replacement of u and v with a single vertex whose incident edges are the edges other than e that were incident to u or v. The resulting graph $G \cdot e$ has one less edge than G.





Contd…

- In a drawing of *G*, contraction of *e* shrinks the edge to a single point, Contracting an edge can produce multiple edges or loops.

- To count spanning trees correctly, we must keep multiple edges .

- In other applications of contraction, the multiple edges may be irrelevant. The recurrence applies for all graphs.

Recurrence

Proposition. Let $\tau(G)$ denote the number of spanning trees of a graph G. If $e \in E(G)$ is not a loop, then

$\tau(G) = \tau(G - e) + \tau(G \cdot e)$ 2.2.8

Proof:

- The spanning trees of G that omit e are precisely the spanning trees of $G - e$.
- To show that G has $\tau(G \cdot e)$ spanning trees containing e we show that contraction of e defines a bijection from the set of spanning trees of G containing e to the set of spanning trees of $G \cdot e$.

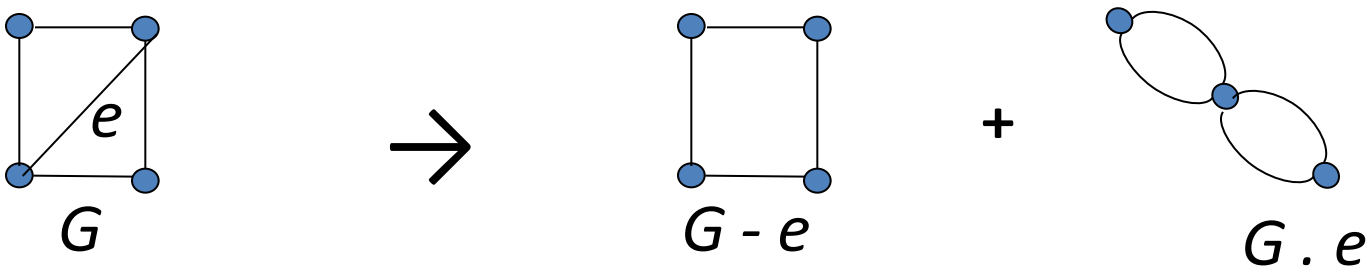
Contd…

Proof:

- When we contract e in a spanning tree that contains e, we obtain a spanning tree of $G \cdot e$, because the resulting subgraph of $G \cdot e$ is spanning and connected and has the right number of edges.
- The other edges maintain their identity under contraction, so no two trees are mapped to the same spanning tree of $G \cdot e$ by this operation. Also, each spanning tree of $G \cdot e$ arises in this way, since expanding the new vertex back into e yields a spanning tree of G. Since each spanning tree of $G \cdot e$ arises exactly once, the function is a bijection.

Example

- A ***step in the recurrence***.
- The graphs on the right each have four spanning trees, so **Proposition 2.2.8** implies that the kite has eight spanning trees, Without the multiple edges, the computation would fail.



- We can save some computation time by recognizing special graphs G where we know $\tau(G)$, such as the graph on the right above.

Remark2.2.10

- If G is a ***connected loopless graph*** with no cycle of length at least 3, then $\tau(G)$ is the product of the edge multiplicities. A disconnected graph has no spanning trees.

- We cannot apply the recurrence of Proposition 2.2.8 when e is a loop. For example, a graph consisting of one vertex and one loop has one spanning tree, but deleting and contracting the loop would count it twice.
- Since loops do not affect the number of spanning trees, we can delete loops as they arise.

Contd…

- Counting trees recursively requires initial conditions for graphs in which all edges are loops.
- Such a graph has one spanning tree if it has only one vertex, and it has no spanning trees if it has more than one vertex.
- If a computer completes the computation by deleting or contracting every edge in a loopless graph G, then it may compute as many as $2^{e(G)}$ **terms**.
- Even with savings from **Remark 2.2.10**, the amount of computation grows exponentially with the size of the graph; this is impractical.

Matrix Tree Theorem (Krichhoff)

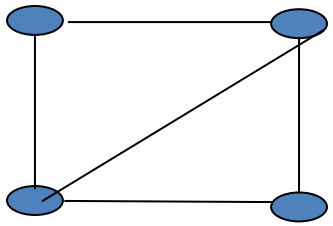
- Another technique leads to a **much faster computation.**

- The **Matrix Tree Theorem**, implicit in the work of **Krichhoff [1847],** computes $\tau(G)$ using a determinant.

- This is much faster, because determinants of *n-by-n* matrices can be computed using fewer than n^3 operations. Also, Cayley's Formula follows from the Matrix Tree Theorem with ***G = Kn*** , but it does not follow easily from Proposition 2.2.8.

Example: A Matrix Tree computation. 2.2.11

- **Theorem 2.2.12** instructs us to form a matrix by putting the vertex degrees on the diagonal and subtracting the adjacency matrix. We then delete a row and a column and take the determinant, When ***G* is the kite of Example 2.2.9**, the vertex degrees are 3,3,2,2. We form the matrix and take the determinant of the matrix in the middle, The result is the number of spanning trees!



G

$$\begin{pmatrix} 3 & -1 & -1 & -1 \\ -1 & 3 & -1 & -1 \\ -1 & -1 & 2 & 0 \\ -1 & -1 & 0 & 2 \end{pmatrix} \rightarrow \begin{pmatrix} 3 & -1 & -1 \\ -1 & 2 & 0 \\ -1 & 0 & 2 \end{pmatrix} \rightarrow 8$$

- Loops don't affect spanning trees, so we delete them before the computation, The proof of the theorem uses properties of determinants.

Matrix Tree Theorem 2.2.12

Theorem: Given a loopless graph G with vertex set $v_1, \ldots, v_n$, let $a_{i,j}$ be the number of edges with endpoints v_i and v_j. Let Q be the matrix in which entry (i, j) is $-a_{i,j}$ when $i \neq j$ and is $d(v_i)$ when $i = j$. If Q^* is a matrix obtained by deleting row s and column t of Q, then $\tau(G) = (-1)^{s+t} \det Q^*$

Conclusion

- In this lecture, we have discussed the Enumeration of trees, Cayley's formula, Prüfer code, Algorithm for generation of Prüfer Sequence from Labelled Tree and Generation of Labelled Tree from Prüfer Sequence, Spanning trees in graphs, Matrix tree computation and Matrix tree theorem.

- In upcoming lecture, we will discuss the minimum spanning tree and shortest paths.