Contraction of an edge

Definition

Let G be a graph and let $e = uv$ be an edge that is not a loop. The contraction of e is the operation that replaces e with a single vertex, which is incident to those edges that were incident to either u or v in G.

Contraction of an edge

Definition

Let G be a graph and let $e = uv$ be an edge that is not a loop. The contraction of e is the operation that replaces e with a single vertex, which is incident to those edges that were incident to either u or v in G.

Denote by $G \cdot e$ the resulting graph.

Contraction of an edge

Definition

Let G be a graph and let $e = uv$ be an edge that is not a loop. The contraction of e is the operation that replaces e with a single vertex, which is incident to those edges that were incident to either u or v in G.

Denote by $G \cdot e$ the resulting graph.

- This construction may create loops and multiple edges.

Contraction of an edge

Definition

Let G be a graph and let $e = uv$ be an edge that is not a loop. The contraction of e is the operation that replaces e with a single vertex, which is incident to those edges that were incident to either u or v in G.

Denote by $G \cdot e$ the resulting graph.

- This construction may create loops and multiple edges.
- $G \cdot e$ has one fewer edge than G.

Deletion-contraction method

How do the numbers $\tau(G)$, $\tau(G - e)$, and $\tau(G \cdot e)$ relate to each other?

Deletion-contraction method

How do the numbers $\tau(G)$, $\tau(G - e)$, and $\tau(G \cdot e)$ relate to each other?

Proposition (Deletion-contraction recurrence)

If $e \in E(G)$ is not a loop, then

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

Deletion-contraction method

How do the numbers $\tau(G)$, $\tau(G-e)$, and $\tau(G \cdot e)$ relate to each other?

Proposition (Deletion-contraction recurrence)

If $e \in E(G)$ is not a loop, then

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

[Example]

Deletion-contraction method

How do the numbers $\tau(G)$, $\tau(G-e)$, and $\tau(G\cdot e)$ relate to each other?

Proposition (Deletion-contraction recurrence)

If $e \in E(G)$ is not a loop, then

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

[Example]

- If e is a loop, one can just delete it since it does not affect the number of spanning trees.

Deletion-contraction method

How do the numbers $\tau(G)$, $\tau(G-e)$, and $\tau(G \cdot e)$ relate to each other?

Proposition (Deletion-contraction recurrence)

If $e \in E(G)$ is not a loop, then

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

[Example]

- If e is a loop, one can just delete it since it does not affect the number of spanning trees.
- With this recurrence, one can in theory compute $\tau(G)$ for any graph recursively, but it's computationally impractical, since one would have to compute up to $2^{e(G)}$ terms.

Matrix Tree Theorem

Theorem

Let G be a loopless graph with vertices $v_1, v_2, \ldots, v_n$.
Let A be its adjacency matrix.

Matrix Tree Theorem

Theorem

Let G be a loopless graph with vertices $v_1, v_2, \ldots, v_n$.
Let A be its adjacency matrix.

Define the matrix

$$Q = \begin{pmatrix} d(v_1) & 0 & \ldots & 0 \\ 0 & d(v_2) & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & d(v_n) \end{pmatrix} - A.$$

Matrix Tree Theorem

Theorem

Let G be a loopless graph with vertices $v_1, v_2, \ldots, v_n$.
Let A be its adjacency matrix.

Define the matrix

$$Q = \begin{pmatrix} d(v_1) & 0 & \ldots & 0 \\ 0 & d(v_2) & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & d(v_n) \end{pmatrix} - A.$$

Let $Q^\star$ be obtained from Q by deleting row s and column t.

Matrix Tree Theorem

Theorem

Let G be a loopless graph with vertices $v_1, v_2, \ldots, v_n$.
Let A be its adjacency matrix.

Define the matrix

$$Q = \begin{pmatrix} d(v_1) & 0 & \ldots & 0 \\ 0 & d(v_2) & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & d(v_n) \end{pmatrix} - A.$$

Let $Q^\star$ be obtained from Q by deleting row s and column t.

Then

$$\tau(G) = (-1)^{s+t} \det Q^\star.$$

Matrix Tree Theorem

Theorem

Let G be a loopless graph with vertices $v_1, v_2, \ldots, v_n$.
Let A be its adjacency matrix.

Define the matrix

$$Q = \begin{pmatrix} d(v_1) & 0 & \ldots & 0 \\ 0 & d(v_2) & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & d(v_n) \end{pmatrix} - A.$$

Let $Q^\star$ be obtained from Q by deleting row s and column t.

Then

$$\tau(G) = (-1)^{s+t} \det Q^\star.$$

[Example] (We won't prove the theorem here.)

Graceful labelings

Definition

A **graceful labeling** of a tree T of order n is a bijection

$$f : V(T) \to \{0, 1, \ldots, n-1\}$$

such that

$$\{|f(u) - f(v)| : uv \in E(T)\} = \{1, 2, \ldots, n-1\}.$$

Graceful labelings

Definition

A **graceful labeling** of a tree T of order n is a bijection

$$f : V(T) \to \{0, 1, \ldots, n-1\}$$

such that

$$\{|f(u) - f(v)| : uv \in E(T)\} = \{1, 2, \ldots, n-1\}.$$

[Example]

Graceful labelings

Definition

A **graceful labeling** of a tree T of order n is a bijection

$$f : V(T) \to \{0, 1, \ldots, n-1\}$$

such that

$$\{|f(u) - f(v)| : uv \in E(T)\} = \{1, 2, \ldots, n-1\}.$$

[Example]

Conjecture (Graceful Tree Conjecture '64)

Every tree has a graceful labeling.

Graceful trees and decompositions

The Graceful Tree Conjecture is known to be true in some special cases.

Graceful trees and decompositions

The Graceful Tree Conjecture is known to be true in some special cases.

Definition

A caterpillar is a tree which contains a path that is incident to every edge.

Graceful trees and decompositions

The Graceful Tree Conjecture is known to be true in some special cases.

Definition

A caterpillar is a tree which contains a path that is incident to every edge.

Exercise (optional): Prove that every caterpillar has a graceful labeling.

2.3 Optimization and trees

A **weighted graph** is a graph with nonnegative numbers on the edges.

2.3 Optimization and trees

A **weighted graph** is a graph with nonnegative numbers on the edges.

They can represent the cost of building a road, or a distance, or the amount of data that can be sent per second.

2.3 Optimization and trees

A **weighted graph** is a graph with nonnegative numbers on the edges.

They can represent the cost of building a road, or a distance, or the amount of data that can be sent per second.

A **minimum spanning tree** is a spanning tree that minimizes the sum of its edge weights.

2.3 Optimization and trees

A **weighted graph** is a graph with nonnegative numbers on the edges.

They can represent the cost of building a road, or a distance, or the amount of data that can be sent per second.

A **minimum spanning tree** is a spanning tree that minimizes the sum of its edge weights.

Minimum Connector Problem: Given an arbitrary weighted connected graph G, find a minimum spanning tree.

Kruskal's algorithm

Input: A weighted connected graph G.

Output: A minimum spanning tree T with edges $e_1, \ldots, e_{n-1}$.

Kruskal's algorithm

Input: A weighted connected graph G.

Output: A minimum spanning tree T with edges $e_1, \dots, e_{n-1}$.

- Start with no edges.

Kruskal's algorithm

Input: A weighted connected graph G.

Output: A minimum spanning tree T with edges $e_1, \ldots, e_{n-1}$.

- Start with no edges.
- At each step, add the edge with smallest weight that does not create a cycle with the edges added so far.

Kruskal's algorithm

Input: A weighted connected graph G.

Output: A minimum spanning tree T with edges $e_1, \ldots, e_{n-1}$.

- Start with no edges.
- At each step, add the edge with smallest weight that does not create a cycle with the edges added so far.
- Finish when we have a spanning tree of G.

Kruskal's algorithm

Input: A weighted connected graph G.

Output: A minimum spanning tree T with edges $e_1, \ldots, e_{n-1}$.

- Start with no edges.
- At each step, add the edge with smallest weight that does not create a cycle with the edges added so far.
- Finish when we have a spanning tree of G.

This is an example of a **greedy algorithm**.

Kruskal's algorithm

Input: A weighted connected graph G.

Output: A minimum spanning tree T with edges $e_1, \ldots, e_{n-1}$.

- Start with no edges.
- At each step, add the edge with smallest weight that does not create a cycle with the edges added so far.
- Finish when we have a spanning tree of G.

This is an example of a **greedy algorithm**.

Theorem

Kruskal's algorithm constructs a minimum-weight spanning tree.