# Planar Graphs, Planarity Testing and Embedding

Swami Sarvottamananda

Ramakrishna Mission Vivekananda University

October 2014

# Outline I



# Outline II



# Introduction

# Scope of the lecture

- *Characterisation of Planar Graphs:* First we introduce planar graphs, and give its characterisation and some simple properties.

# Scope of the lecture

- *Characterisation of Planar Graphs:* First we introduce planar graphs, and give its characterisation and some simple properties.
- *Planarity Testing:* Next, we give an algorithm to test planarity.

# Scope of the lecture

- *Characterisation of Planar Graphs:* First we introduce planar graphs, and give its characterisation and some simple properties.
- *Planarity Testing:* Next, we give an algorithm to test planarity.
- *Planar Embedding:* Lastly we see how a given planar graph can be embedded in a plane.
  We also explore straight line embeddings.

# What is a Drawing?

## Definition (Drawing)

Given a graph $G = (V, E)$, a drawing maps each vertex $v \in V$ to a distinct point $\Gamma(v)$ in plane, and each edge $e \in E$, $e = (u, v)$ to a simple open jordan curve $\Gamma(u, v)$ with end points $\Gamma(u)$, $\Gamma(v)$.

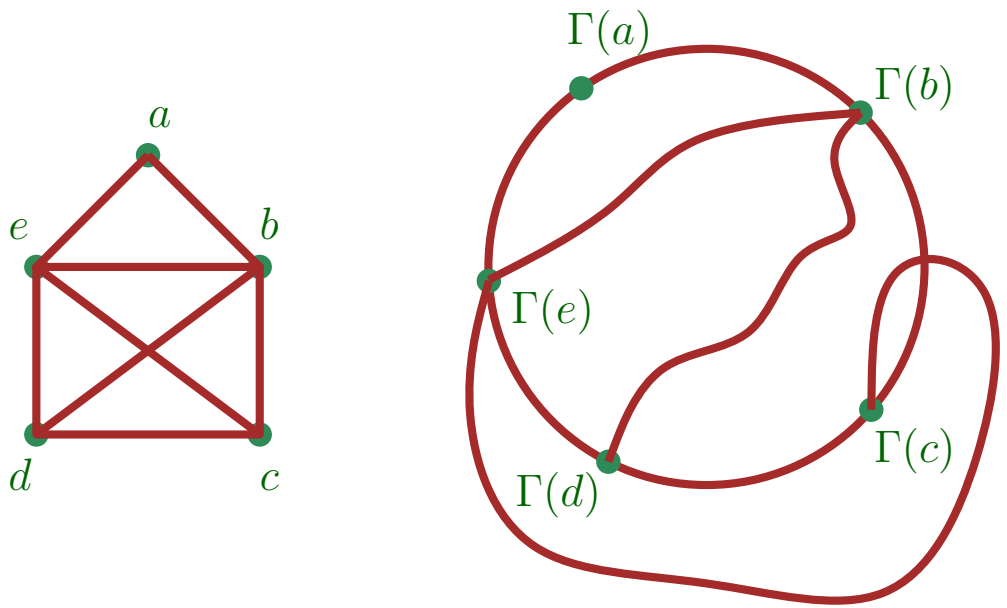



Figure: Drawing of a graph

# What is a Planar Graph?

## Definition (Planar Graphs)

Given a graph $G = (V, E)$, $G$ is planar if it admits a drawing such that no two distinct drawn edges intersect except at end points.

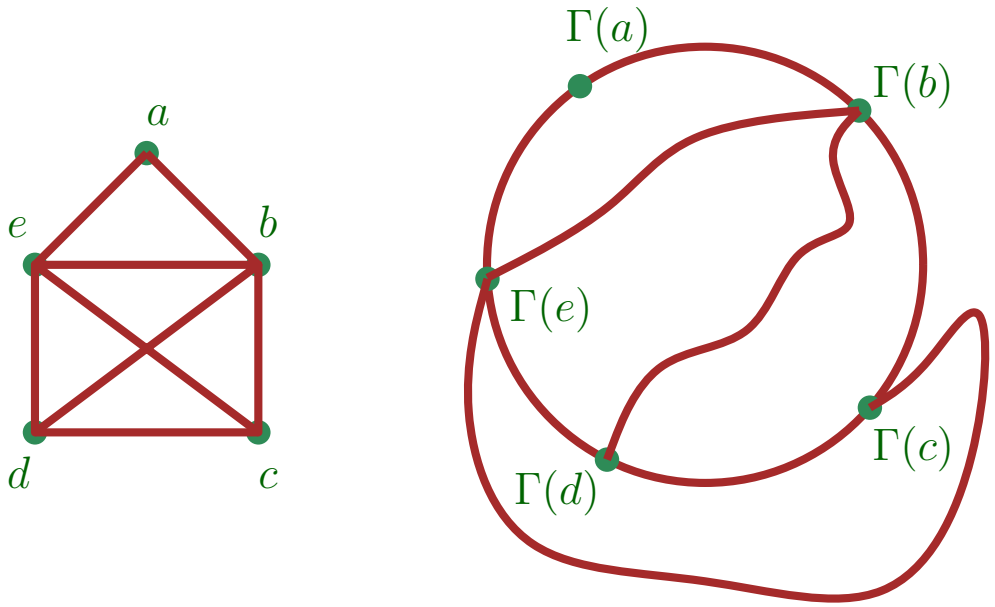



Figure: Planar drawing of a graph

# Motivation

# Properties of Planar Graphs

There are number of interesting properties of planar graphs.

- They are sparse. There size including faces, edges and vertices is O(n).
- They are 4-colourable.
- A number of operations can be performed on them very efficiently. Since there is a topological order to the incidences.
- They can be efficiently stored (A data structure called $SPQR$-trees even allows $O(1)$ flipping of planar embeddings).

# Applications of Planar Graphs

Planar graphs are extensively used in Electrical and Civil engineering.

- Easy to visualize. In fact, crossings reduces comprehensibility. So all good graph drawing tools use planar graphs.
- VLSI design, circuit needs to be on surface: lesser the crossings, better is the design.
- Highspeed Highways/Railroads design, crossings are always problematic.
- Irrigation canals, crossings simply not admissible.
- Determination of *isomorphism* of chemical structures.
- Most of facility location problems on maps are actually problems of planar graphs.

# Problem Definition

# Problem Definition: Planarity Testing

**Problem (Decision Problem)**

*Given a graph $G = (V, E)$, is $G$ planar, i.e., can $G$ be drawn in the plane without edge crossings?*

# Problem Definition: Planarity Embedding

**Problem (Computation Problem)**

*Given a graph $G = (V, E)$, if $G$ is planar, how can $G$ be drawn in the plane such that there are no edge crossings? I.e., compute a planar representation of the graph $G$.*

# Question: $K_4$?

Is the following graph ($K_4$) planar?

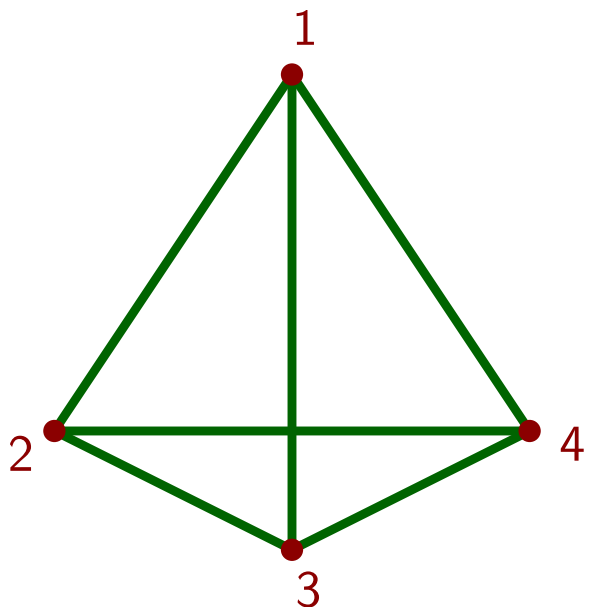



Figure: Graph $K_4$

# Answer: $K_4$ is planar

Yes, $K_4$ is planar!

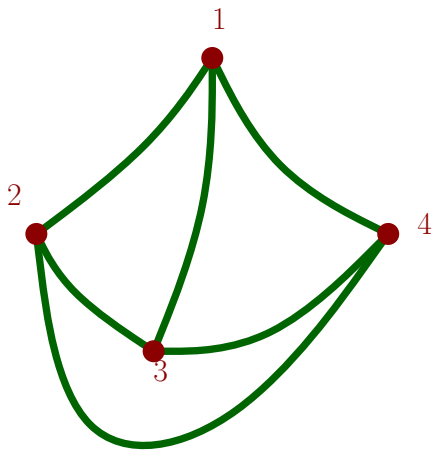



Figure: Planar $K_4$

# Question: $K_5$ and $K_{3,3}$?

Are the following graphs ($K_5$ and $K_{3,3}$) planar?

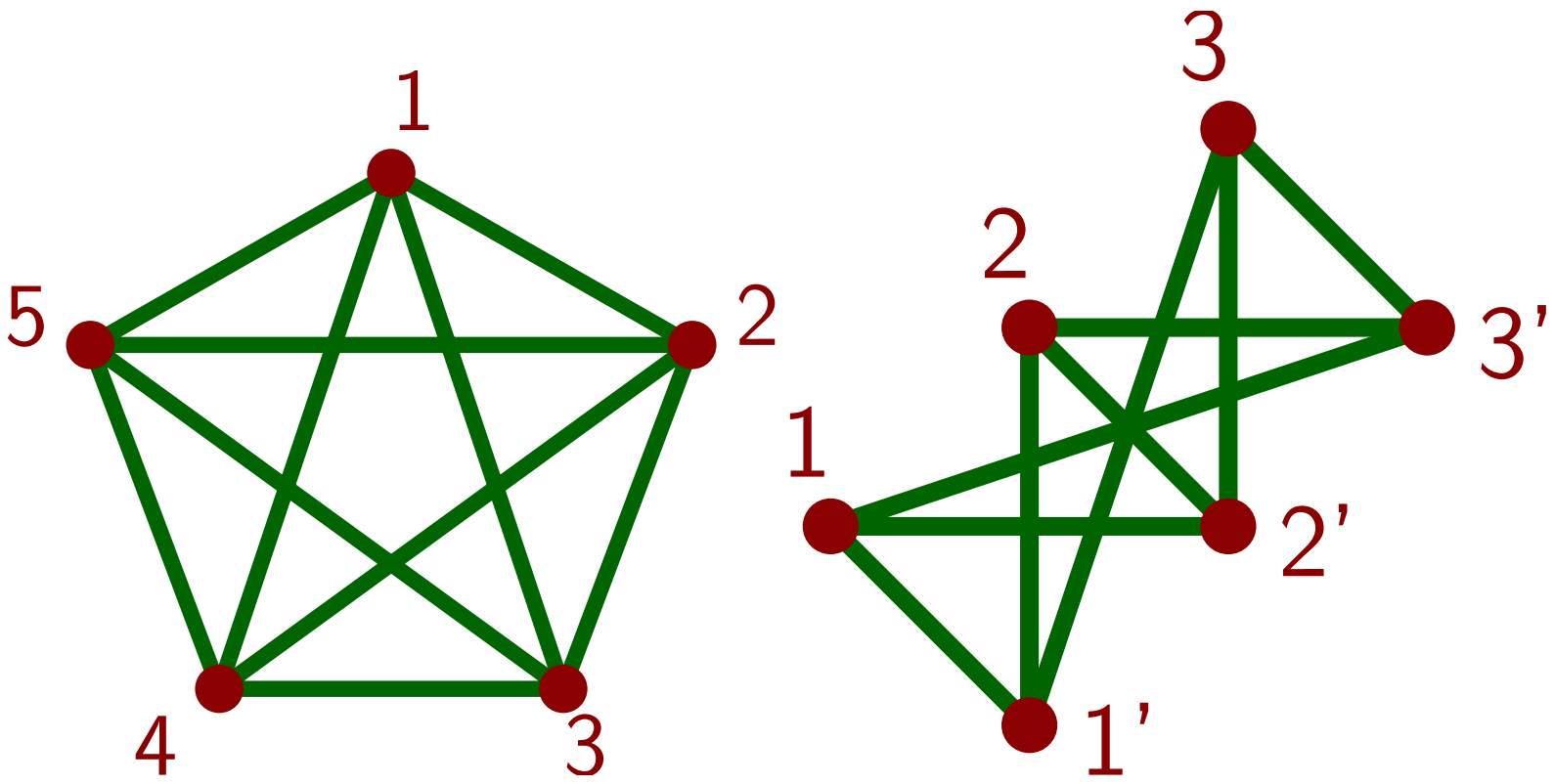



Figure: Graphs $K_5$ and $K_{3,3}$

# Answer: $K_4$ is planar

No!! They aren't. There always will be at least one crossing.

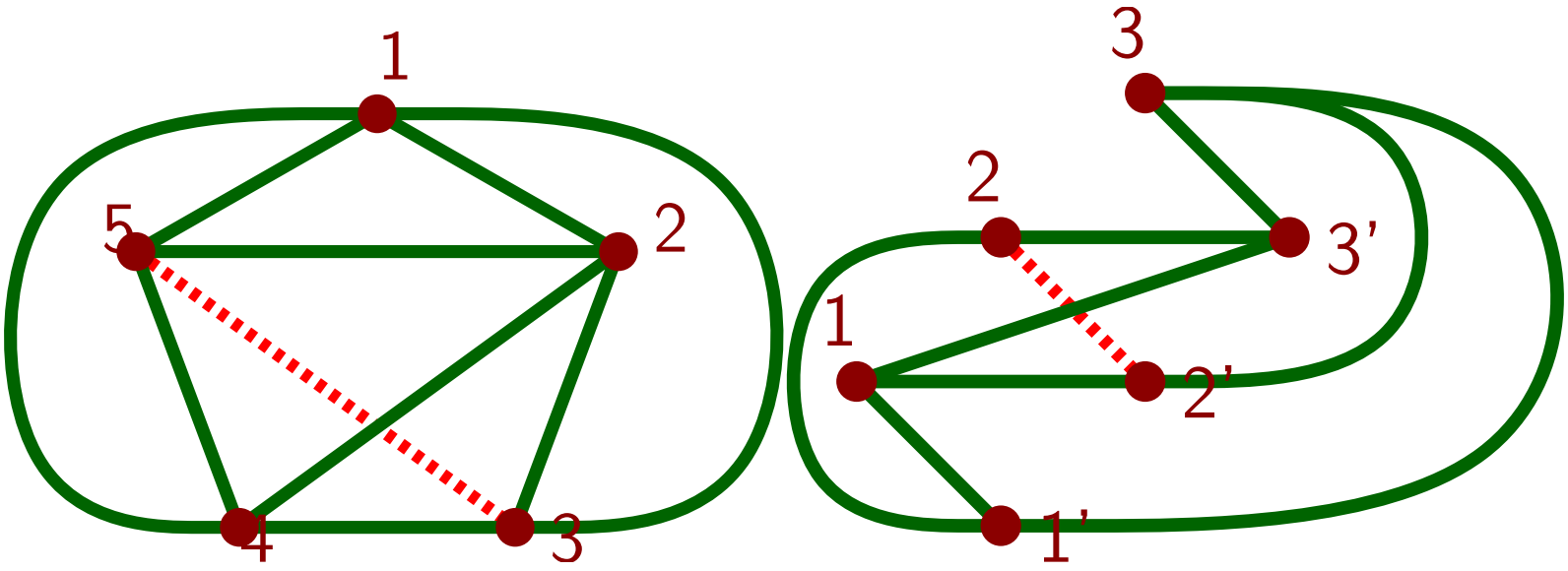



Figure: Non-planarity of $K_5$ and $K_{3,3}$

Full proofs by Euler's celebrated theorem.

# Question: Is a given graph planar?

Is the following graph planar?
There are a lot of crossings $O(n^2)$.

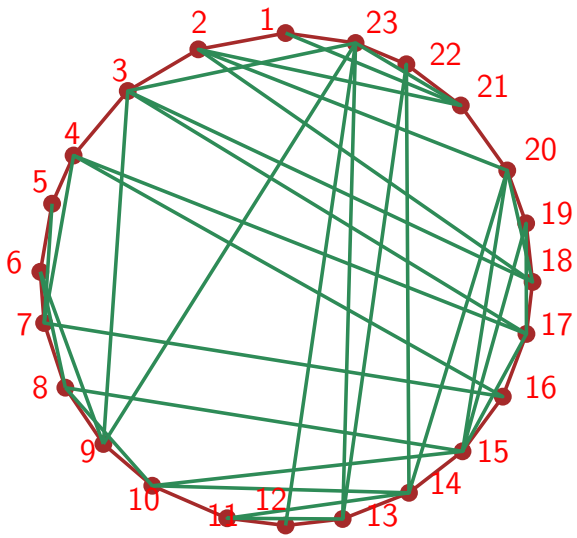



Figure: A Hamiltonian Graph

# Answer: Yes

Yes, it is.

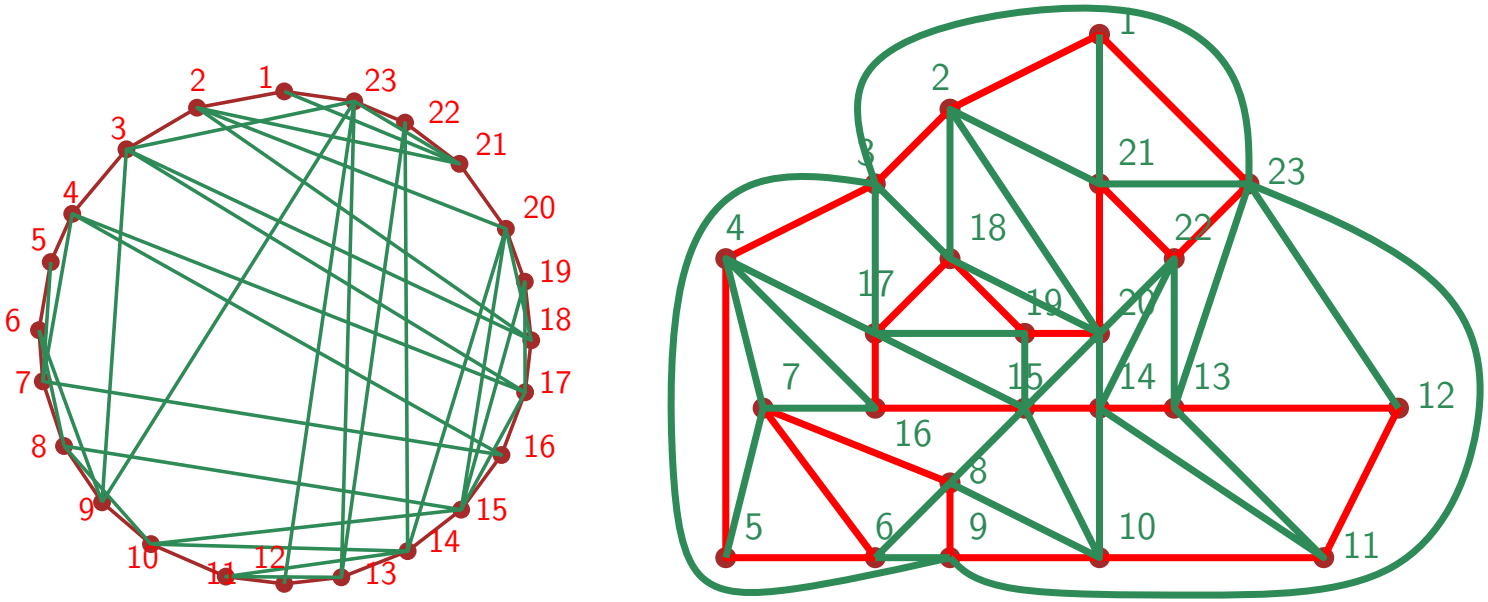



Figure: Planar embedding of the given graph

But, how to arrive at this answer? It is tough.

# Characterisation of Planar Graphs

# Basic Assumptions

Euler's formula gives the necessary condition for a graph to be planar[3].

We assume that our graphs are connected and there are no self loops and no multi-edges.

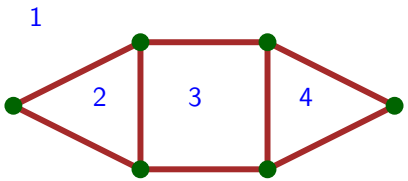
Disconnected graphs, 1-degree vertices, multi-edges can be easily dealt with.

# Euler's Relation for nodes, arcs and faces[1]

## Theorem (Euler's Relation)

*Given a planar graph with n vertices, m edges and f faces:*

$$n - m + f = 2$$




A Planar Graph :
$n = |V| = 6$,
$m = |E| = 8$, and
$f = |F| = 4$
$6 - 8 + 4 = 2$

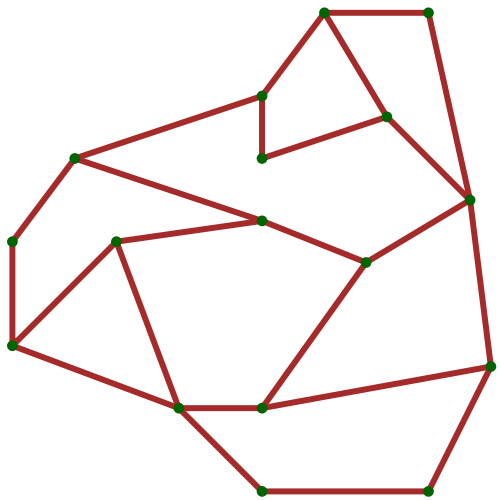
[1] The exterior is also counted as a face. The above relation also applies to simple polyhedrons with no holes.

# Euler's Relation for nodes, arcs and faces

**Theorem (Euler's Relation)**

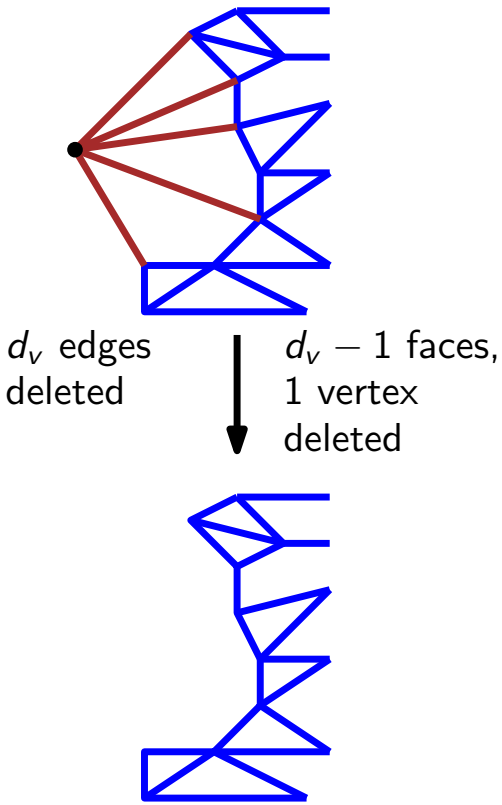
*Given a planar graph with n vertices, m edges and f faces:*

$$n - m + f = 2$$



A Planar Graph :
$n = |V| = 17$,
$m = |E| = 24$, and
$f = |F| = 9$
$17 - 24 + 9 = 2$

# Euler's Relation for nodes, arcs and faces




*Proof:* We prove Euler's Relation by Mathematical Induction on vertices.

If we remove one vertex $v$ on boundary (don't delete a cut vertex) with $d_v$ edges, then we are removing $d_v - 1$ faces.

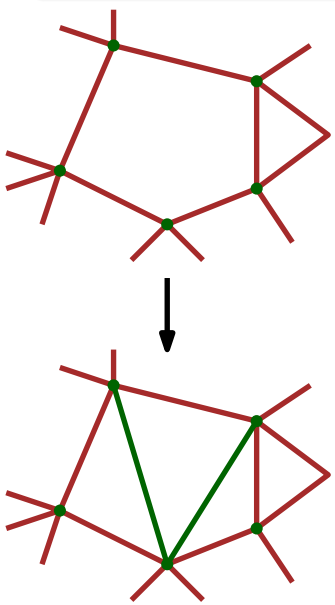
Thus the invariant $n - m + f = 2$ is maintained.

Basis is an isolated vertex, $n = f = 1, m = 0$.□

# Euler's Relation: Corollary 1

## Corollary

*For a maximal planar graph, where each face is a triangle, $m = 3n - 6$, and therefore, for any planar graph with at least three vertices, we should have: $m \leq 3n - 6$.*



*Proof:* $\sum_{f \in F} e_f = 2m$ and therefore since

$e_f \geq 3 \quad \Rightarrow \quad \sum_{f \in F} e_f \geq 3f \quad \Rightarrow$
$2m \geq 3f.$

Substituting in $n - m + f = 2$ gives us $n - m + 2m/3 \geq 2$ $\square$

# Euler's Relation: Corollary 1

Example to show $m \leq 3n - 6$.

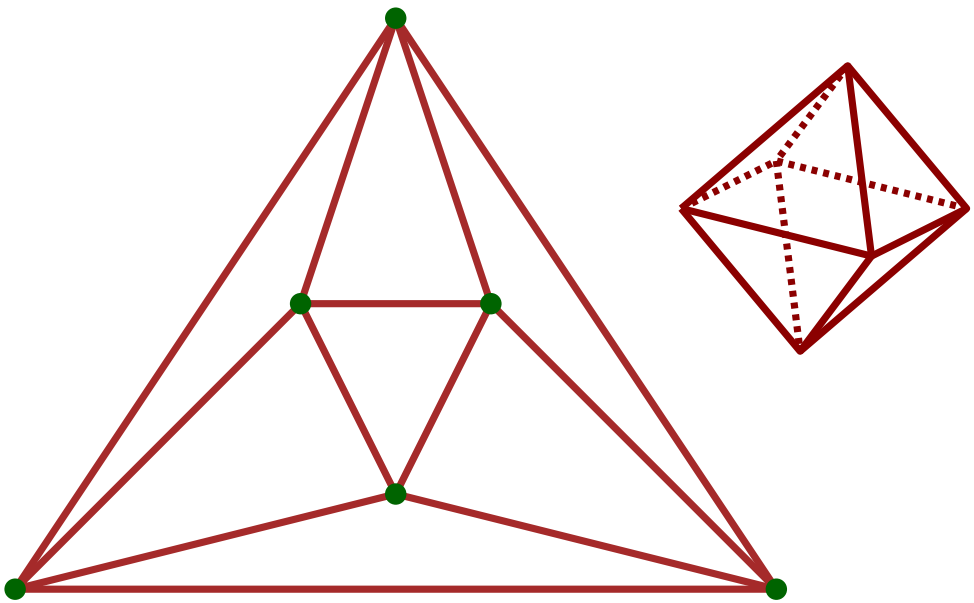


Figure: Octahedron: $n = 6, m = 12,\ m \leq 3n - 6$

# Non-planarity of $K_5$

Earlier we said that $K_5$ is non-planar.

**Lemma**

*$K_5$ is non-planar.*

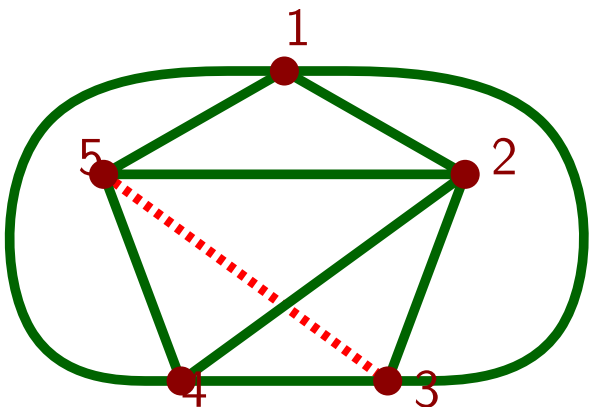



Figure: *Proof:* $n = 5, m = 10,\ m > 3n - 6\ (= 9)$ $\square$

# Euler's Relation: Corollary 2

**Corollary**

*For a planar graph, where no face is a triangle:* $m \leq 2n - 4$.

*Proof:* $\sum_{f \in F} e_f = 2m$ and therefore since

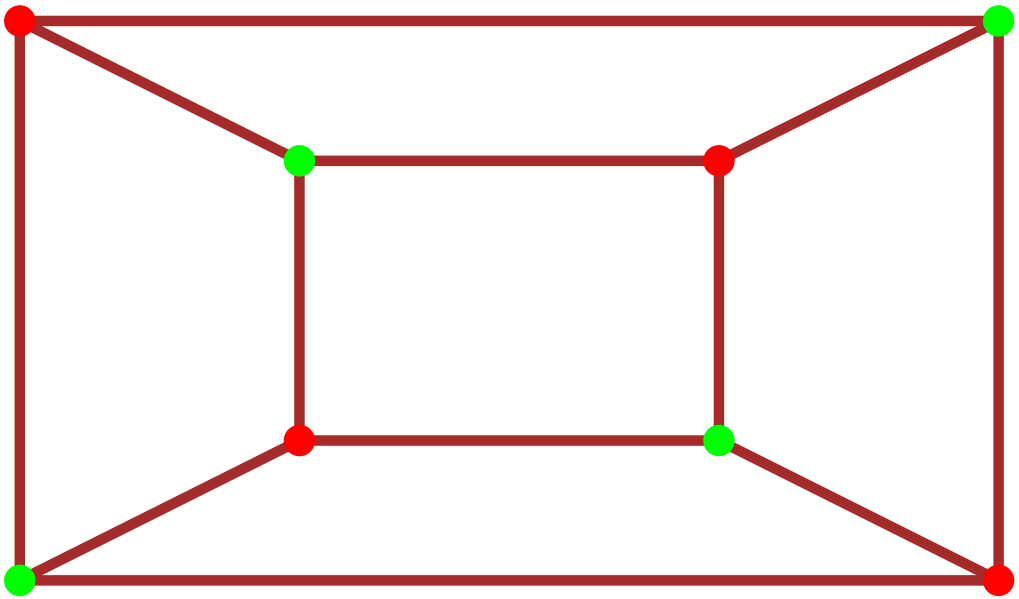
$e_f \geq 4 \quad \Rightarrow \quad \sum_{f \in F} e_f \geq 4f \quad \Rightarrow \quad 2m \geq 4f.$

Substituting in $n - m + f = 2$ gives us $n - m + m/2 \geq 2$ $\square$

Above relation is true for bi-partite planar graphs and graphs with no 3-cycle.

# Euler's Relation: Corollary 2

Example to show $m \leq 2n - 4$ for graphs without triangles.



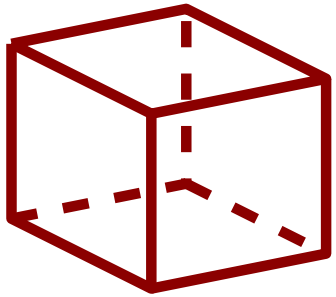


Figure: Cube : $n = 8, m = 12,\ m \leq 2n - 4$

# Non-planarity of $K_{3,3}$

Earlier we said that $K_{3,3}$ is non-planar.

**Lemma**

*$K_{3,3}$ is non-planar.*

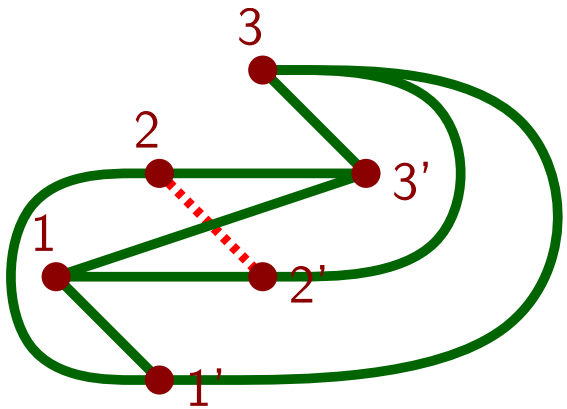
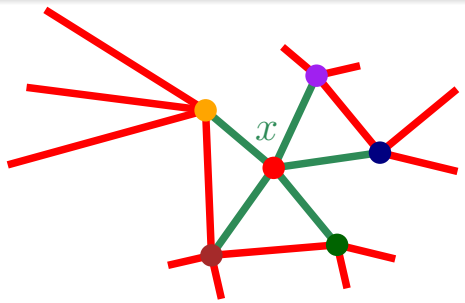



Figure: *Proof:* $n = 6, m = 9,\ m > 2n - 4\ (= 8)$ $\square$

# Euler's Relation: Corollary 3

**Corollary**

*Any planar graph is 6 colourable.*




*Proof:*

- Since $m \leq 3n - 6$, there exists a vertex with degree less than 6 (otherwise $\sum_v d_v = 2m \Rightarrow 2m \geq 6n$).
- By induction, if we remove this vertex, resulting graph is 6-colourable.
- Just give this vertex a colour other than the five colours of the neighbours.

□

# Euler's Relation: Corollary 4

**Corollary**

*Any planar graph is 5 colourable.*

# 5-Colorability of planar graphs

*Proof:*

- The neighbours of a 5-degree vertex aren't all connected.
- Take one such pair and do the following:

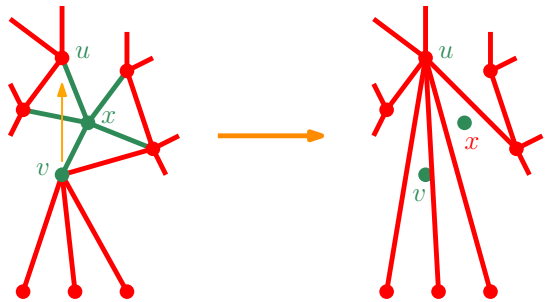



Figure: $u$ and $v$ are not connected

- By induction, if we remove the vertex $x$ and $v$, resulting graph is 5-colourable.
- Just give $v$ same colour as $u$, and $x$ a colour other than the 4 colours of the neighbours.

□

# Insufficiency of Euler's Conditions

Euler's conditions are necessary but not sufficient, for example join $K_5$ and $K_{3,3}$ by an edge.

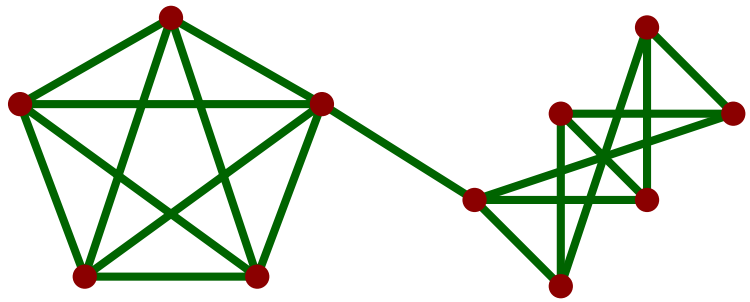


Figure: $n = 11, m = 20, m \leq 3n - 6$

However, above graph is non-planar.

# Kurotowski's and Wagner's Theorem for Planar Graphs

# Kuratowski's and Wagner's Theorems

Next we look at Kuratowski's and Wagner's Theorems for conditions of sufficiency.

Before stating the theorems, we need to understand subdivisions and minors of a graph.

# What are subdivisions and minors of a graph?

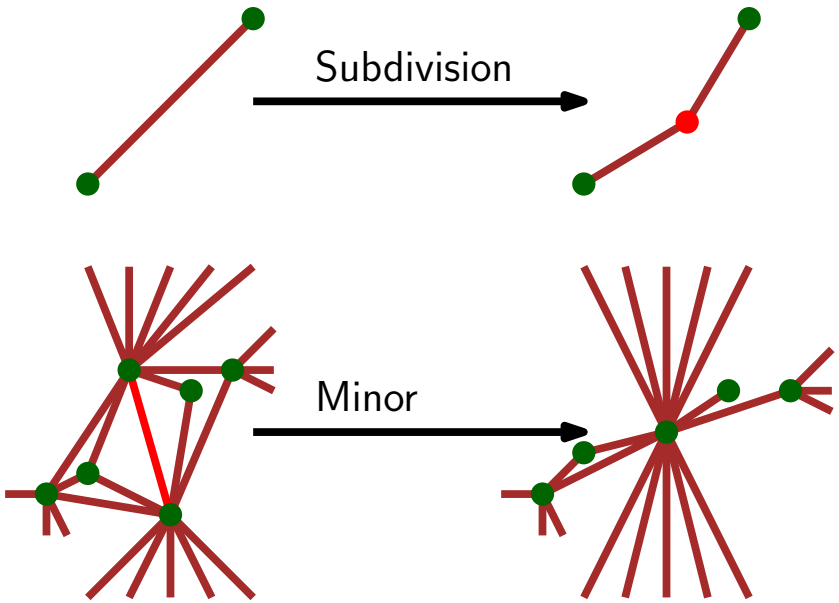



Figure: Subdivisions and Minors

- Subdividing an edge in a planar (non-planar) graph does not make it non-planar (planar).
- Shrinking an edge of a planar graph to make a single vertex does not make it non-planar.

# What are subdivisions and minors of a graph?

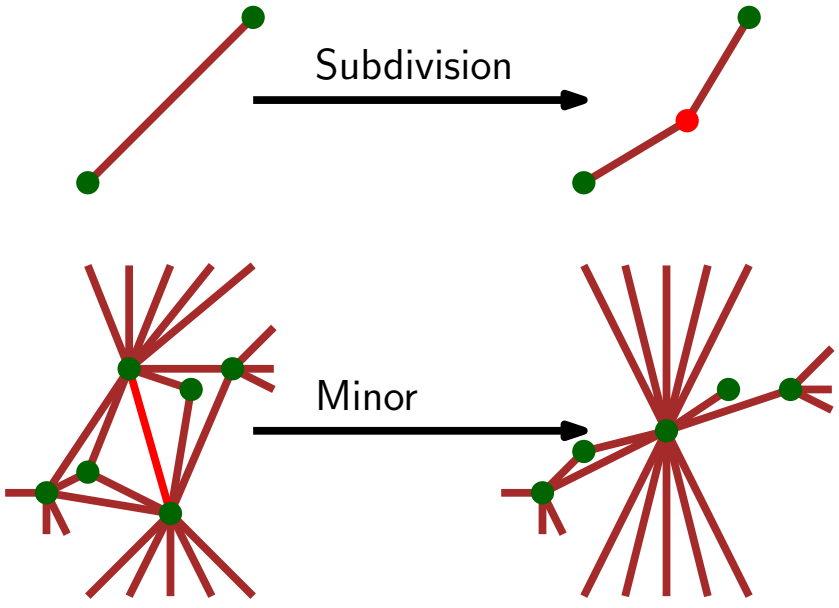



Figure: Subdivisions and Minors

In a way subdivision and minors are complementary. In subdivisions we add a vertex and in minors we remove a vertex.
A graph is planar iff its subdivision is planar.
However, if a graph is planar then its minor is planar.

# Kuratowski's Theorem

**Fact**

*Subdividing any edge means replacing the edge with a path of length 2.*

**Theorem (Kuratowski's Theorem[6])**

*$G$ is planar iff $G$ contains no sub-division of $K_5$ or $K_{3,3}$.*

As noted earlier, subdividing an edge in a planar graph does not make it non-planar.

# Wagner's Theorem

An alternate characterisation of planar graphs by Wagner[10].

**Theorem (Wagner's Theorem)**

*$G$ is planar iff $G$ contains no subgraph which has $K_5$ or $K_{3,3}$ as minor.*

See [4] for yet another characterisation by Harary and Tutte.
As noted earlier, shrinking an edge of a planar graph to make a single vertex does not make it non-planar.

# Necessity of Subdivisions or Minors

Can we do away with subdivisions or minors?

- Algorithmically, checking all minors or subdivisions is very expensive for planarity testing. Exponential if checked naively.
- Checking $K_5$ or $K_{3,3}$ as subgraphs is much efficient, at least it is polynomial.
- Why do we need subdivisions or minors? Aren't subgraphs sufficient?
- We show a non-planar graph which does not have a $K_5$ or $K_{3,3}$ as a subgraph. So, answer is no.

# Peterson's Graph

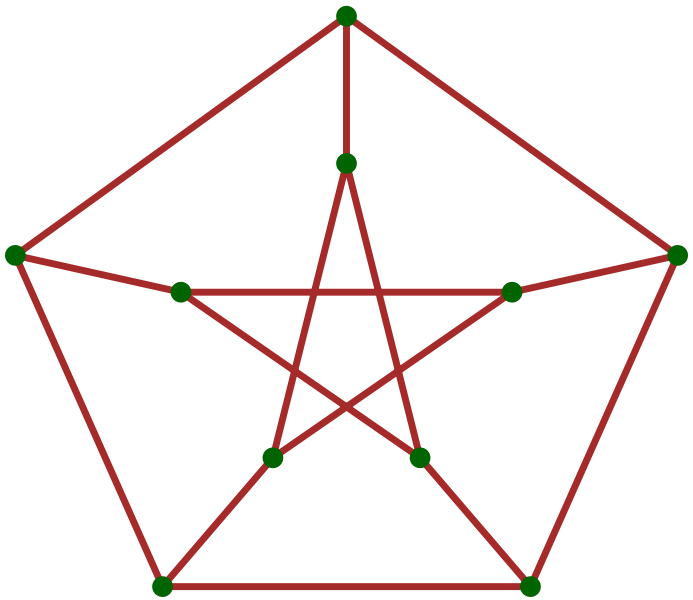


Figure: Peterson's Graph

A graph which doesn't have $K_5$ or $K_{3,3}$ as a subgraph. However, it has a subdivision of $K_{3,3}$ and both $K_5$ and $K_{3,3}$ as minors.

# Non-planarity of Peterson's Graph

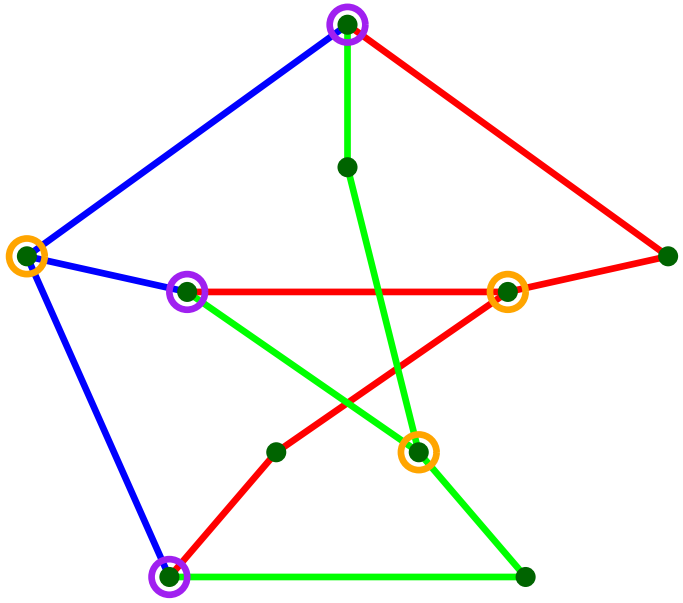


Figure: Kuratowski's theorem applied to the peterson's graph

Peterson Graph has a subdivision of $K_{3,3}$ . The same subgraph can be shrunk to get $K_{3,3}$ as minor.

# Non-planarity of Peterson's Graph

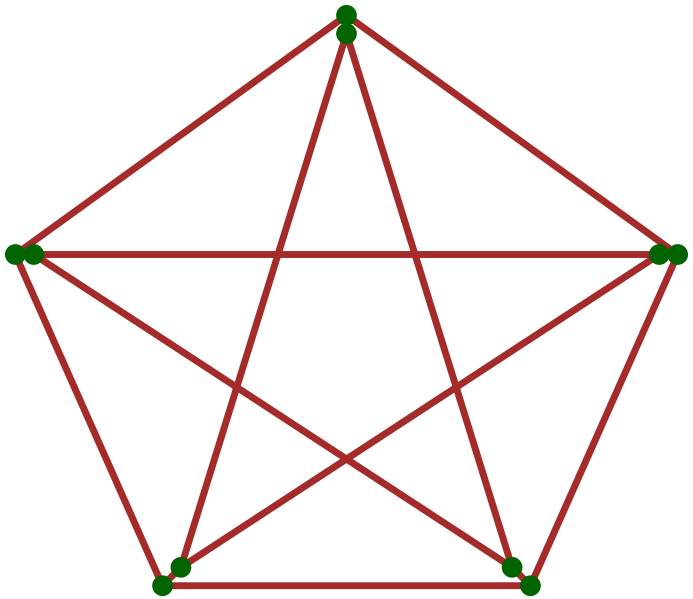
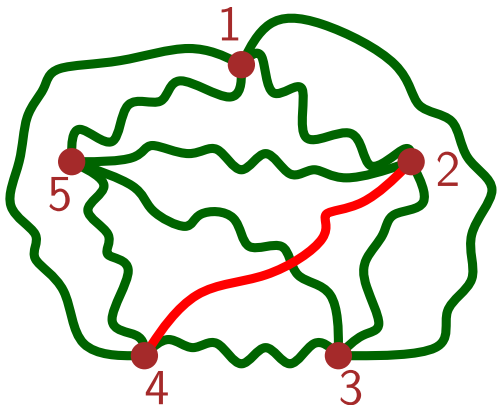


Figure: Wagner's theorem applied to the peterson's graph

Peterson Graph has $K_5$ as a minor.

# Necessity Proof of Kuratowski's Theorem




*Proof:* (of necessity) Necessity immediately follows from non-planarity of $K_5$ and $K_{3,3}$.

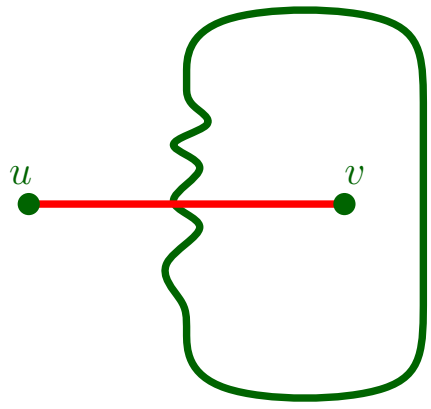
Any subdivison of $K_5$ and $K_{3,3}$ is also non-planar. □

Necessity condition of Wagner's theorem can also be proved easily. Shrinking edges will not change planarity. So, if we get $K_5$ or $K_{3,3}$ by shrinking edges, then initial subgraph must be non-planar to start with.

# Sufficiency proof of Kuratowski's Theorem

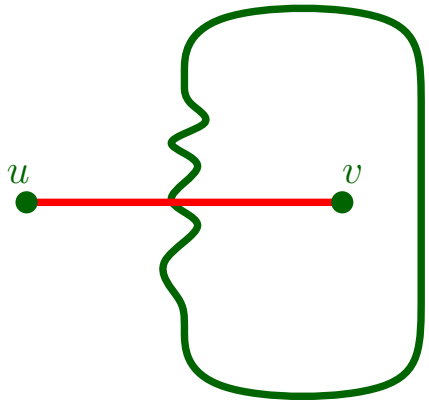
*Proof:* (of sufficiency) Proof of sufficiency is a little difficult.

- Suppose G is non-planar.
- First we remove edges and vertices of a non-planar graph such that it becomes a minimal non-planar graph. I.e. removing any edge will make the resulting graph planar.
- We ask: How does removing an edge of a non-planar graph make it planar?




- Somehow we always need to join the vertices for the last edge, inside and outside of a bad cycle.

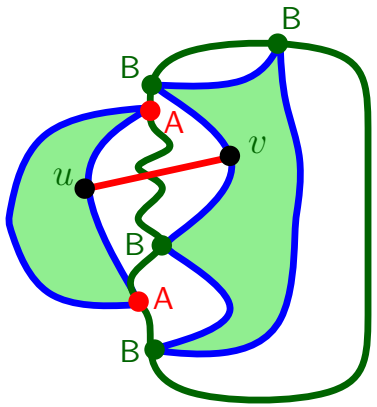
# Sufficiency proof of Kuratowski's Theorem (contd.)




Why can't we move $u$ or $v$ to the other side?

This cycle must be such that $u$ and $v$ are always on the opposite sides. There must be something which is stopping us bringing to the same side.

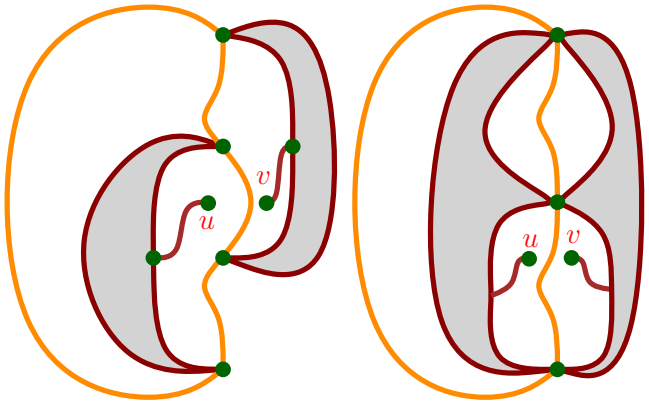
# Sufficiency proof of Kuratowski's Theorem (contd.)




Components (minus cycle) containing $u$ and $v$ are conflicting with each other.
Conflict only depends on placement of attachments.

# Sufficiency proof of Kuratowski's Theorem (contd.)

All the conflicting cases can be brought under !! a finite number of cases !!.

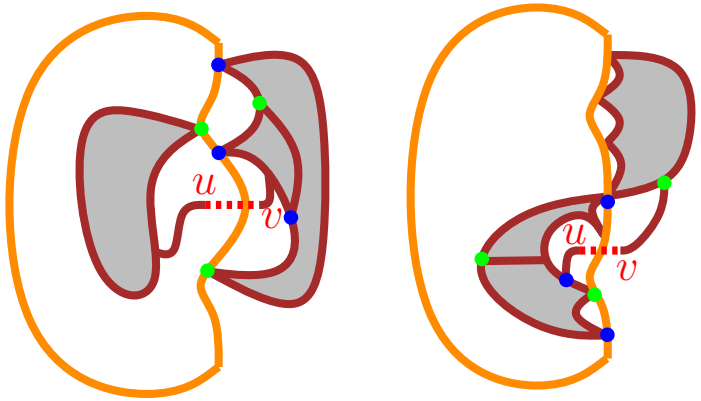



- Both component has a pair of attachments that has uncommon attachments which are alternating.
- Both components have all common attachments, which number more than three.

The dangling path to $u$ and $v$ may be empty.

# Sufficiency proof of Kuratowski's Theorem (contd.)

These are another bunch of two cases




- One or both of the components have only one attachment, but $u$ connects to $v$ of non-enclosing attachments.
- All uncommon attachment pairs are non-alternating with at most one common attachment in between, but $v$ connects to $u$ in between non-alternating attachments.

The dangling path to $u$ and $v$ may be empty. We also need to have some additional paths.

# Sufficiency proof of Kuratowski's Theorem (contd.)

What if the components aren't like what was shown?

Then we can bring both the components on the same side and look for another cycle.

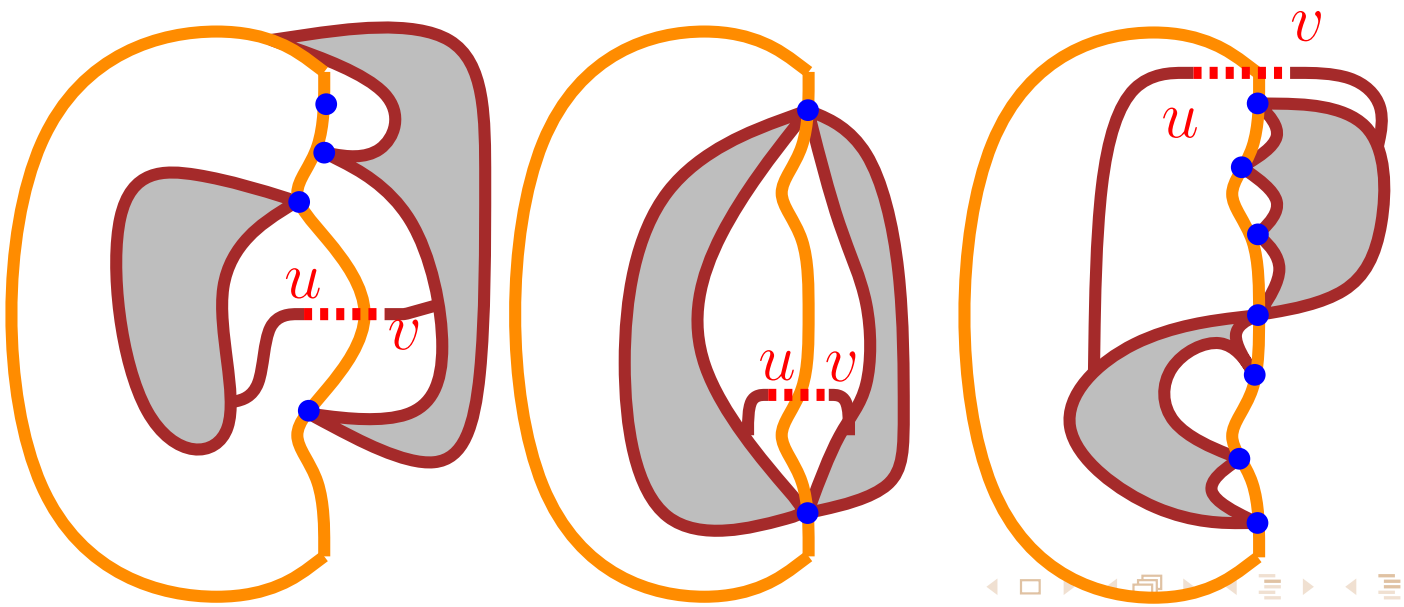
We repeat until we get to the above cases. At every step we decrease at least one crossing.

We show the other cases in the next slide where we can switch sides.

# Sufficiency proof of Kuratowski's Theorem (contd.)

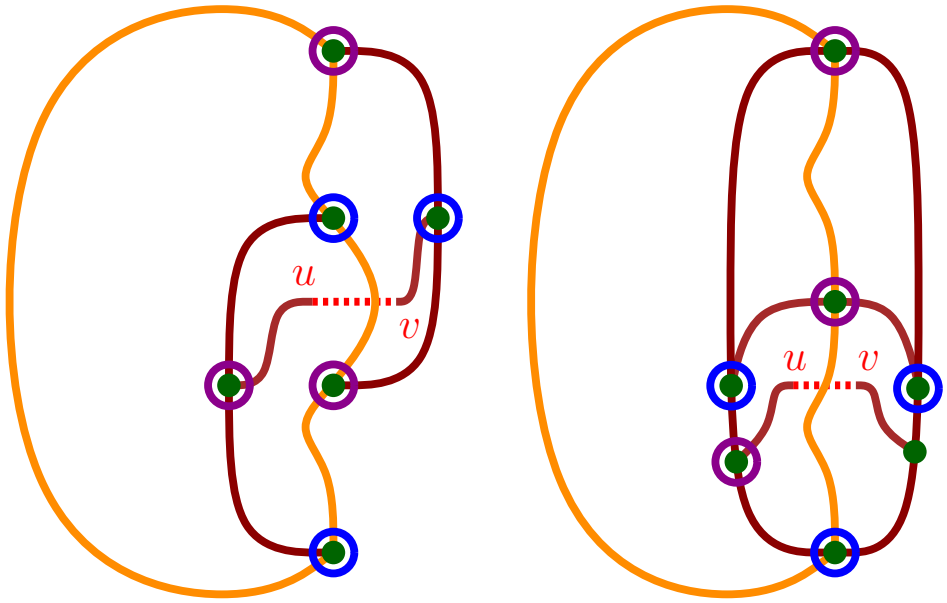
A few other non-conflicting cases are (for the sake of fuller argument):

- One or both of the components have only one attachment and $uv$ as below.
- There are only two common attachments and
- All uncommon attachment pairs are non-alternating with at most one common attachment in between with $uv$ as below.



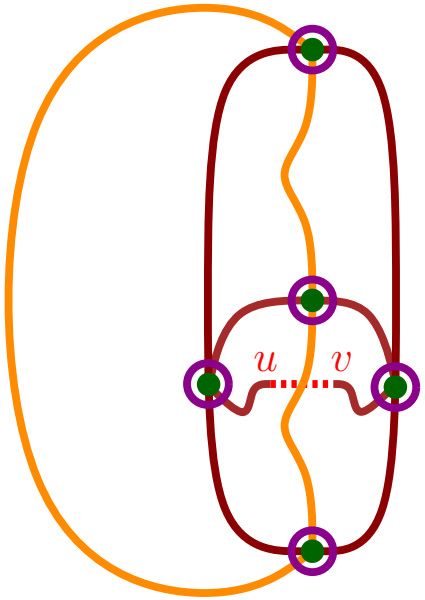

# Sufficiency proof of Kuratowski's Theorem (contd.)

So, in the conflicting cases, we get:




Either there is a pair of uncommon alternating attachments which conflict or common triplet of conflicting attachments as above (with at least one dangling path not at junction for triplet case). We get $K_{3,3}$'s subdivision.

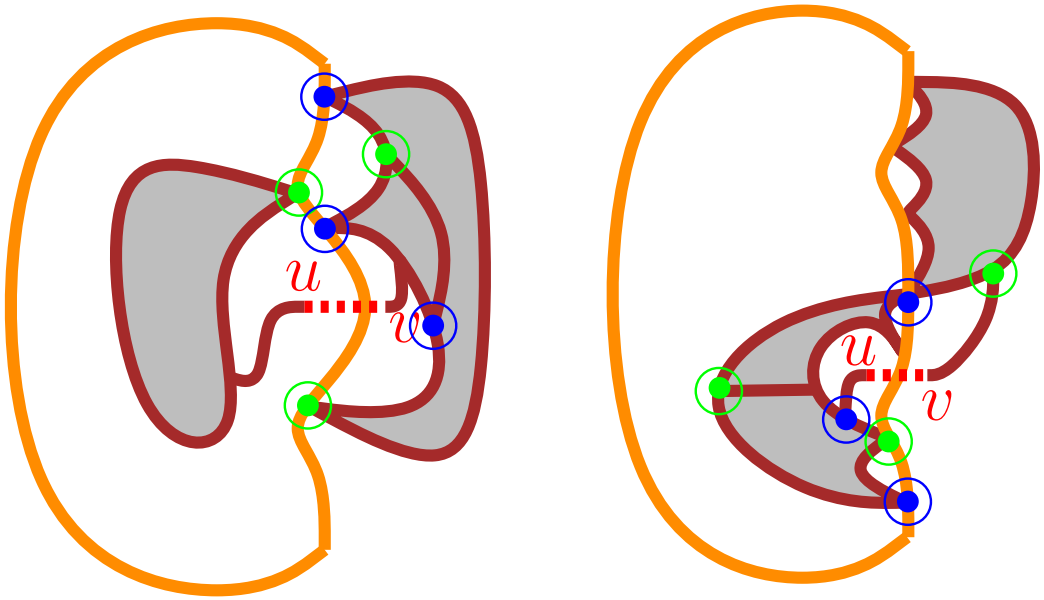
# Sufficiency proof of Kuratowski's Theorem (contd.)




Or, there are three common attachments but which make conflicting attachments as above. The dangling paths are at junction. We get $K_5$'s subdivision.

# Sufficiency proof of Kuratowski's Theorem (contd.)

For the other two cases mentioned, I show only single crossings of $uv$, multiple are similar.




We get $K_{3,3}$'s subdivision.

# Sufficiency proof of Kuratowski's Theorem (contd.)

Thus it can be shown that

**Lemma**

*If $G$ is non-planar it must contain either a sub-division of $K_{3,3}$ or $K_5$.*

# Planarity Testing of Graphs

# Testing Planarity using Kuratowski's Theorem

Question is—How to apply Kuratowski's theorem?

A direct application is an exponential method.

# Testing Planarity using Kuratowski's Theorem (contd.)

To test for $K_5$'s subdivision:

- Choose 5 vertices of $G$. There are $\binom{n}{5}$ choices.
- Check if all 5 vertices are connected by $\binom{5}{4} = 10$ distinct paths as in $K_5$. No. of paths is exponential.

# Testing Planarity using Kuratowski's Theorem (contd.)

To test for $K_5$'s subdivision:

- Choose 5 vertices of $G$. There are $\binom{n}{5}$ choices.
- Check if all 5 vertices are connected by $\binom{5}{4} = 10$ distinct paths as in $K_5$. No. of paths is exponential.

To test for $K_{3,3}$'s subdivision:

- Choose 6 vertices of $G$. There are $\binom{n}{6}$ choices.
- Check if 6 vertices are connected by $3 \times 3 = 9$ distinct paths as in $K_{3,3}$. No. of paths is exponential.

# Testing Planarity using Kuratowski's Theorem (contd.)

To test for $K_5$'s subdivision:

- Choose 5 vertices of $G$. There are $\binom{n}{5}$ choices.
- Check if all 5 vertices are connected by $\binom{5}{4} = 10$ distinct paths as in $K_5$. No. of paths is exponential.

To test for $K_{3,3}$'s subdivision:

- Choose 6 vertices of $G$. There are $\binom{n}{6}$ choices.
- Check if 6 vertices are connected by $3 \times 3 = 9$ distinct paths as in $K_{3,3}$. No. of paths is exponential.

Hard to see how both can be done in sub-exponential time.

# Testing Planarity using Wagner's Theorem

To test for a $K_5$ or $K_{3,3}$ minor:

- Choose an edge of $G$ ($m$ choices).
- Shrink it.
- If 6 vertices are remaining check for $K_{3,3}$.
- If 5 vertices are remaining check for $K_5$.
- Repeat.

# Testing Planarity using Wagner's Theorem

To test for a $K_5$ or $K_{3,3}$ minor:

- Choose an edge of $G$ ($m$ choices).
- Shrink it.
- If 6 vertices are remaining check for $K_{3,3}$.
- If 5 vertices are remaining check for $K_5$.
- Repeat.

This is easier to understand. However, worst case is still $O(m!)$.

# Kurotowski's and Wagner's Conditions are not useful

Conclusion: both are obviously exponential time algorithms.

How can we do it more efficiently?

# Hopcroft and Tarjan's Planarity Testing Algorithm
# &
# Planar Embedding by Mehlhorn and Mutzel[5, 7, 8]

# Basic Assumption

We assume the given graph is connected.

We further assume the given graph is bi-connected.

We get to biconnectivity in a moment.

# Preliminaries

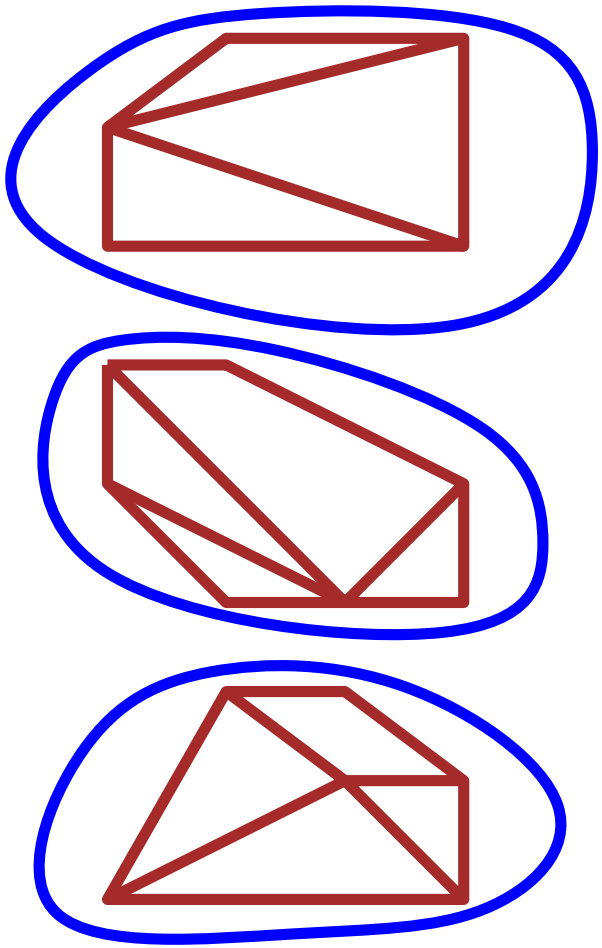


Figure: Disconnected Components

**Fact**

*Disconnected components can be tested for planarity separately.*

# Basic Assumptions-II

What is a biconnected or 2-connected graph?

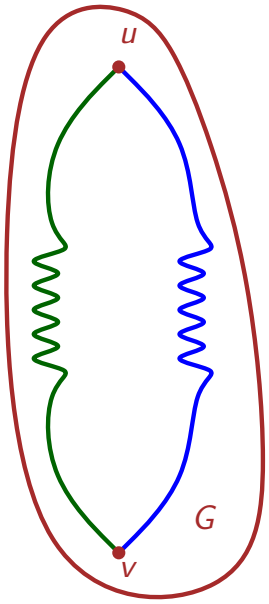


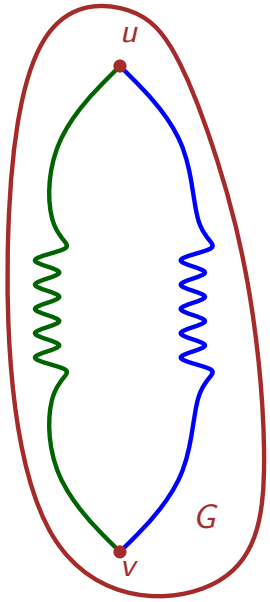

Figure: Biconnected Graph

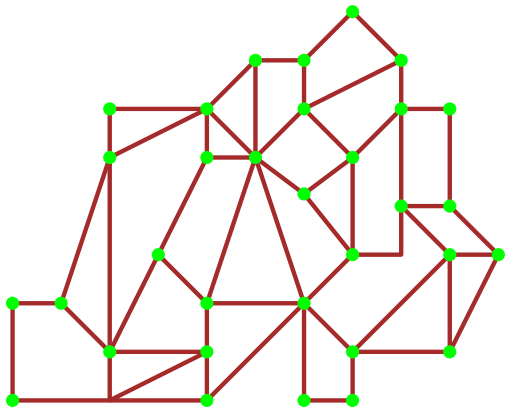
**Definition**

A graph $G$ is biconnected iff every pair of vertices $u$ and $v$ are connected by at least two vertex disjoint paths except at $u$ and $v$, which are end points.

# Basic Assumptions-II

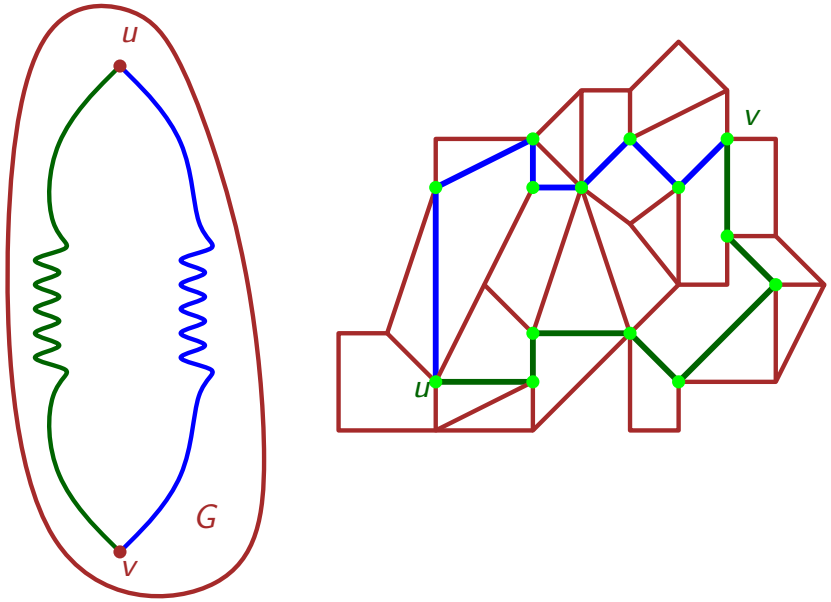
Is the following graph bi-connected?



# Basic Assumptions-II

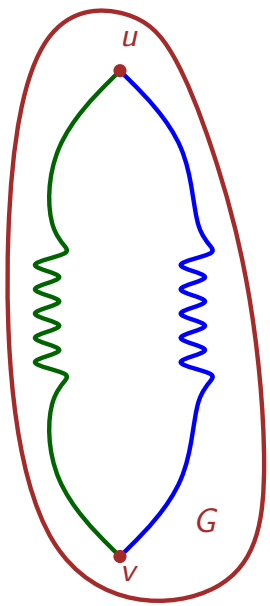
Is the following graph bi-connected?

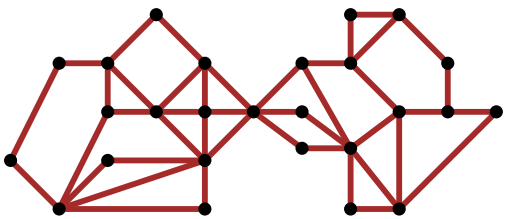



Yes! Above graph has every pair of vertices connected to each other by two vertex disjoint paths.

# Basic Assumptions-II

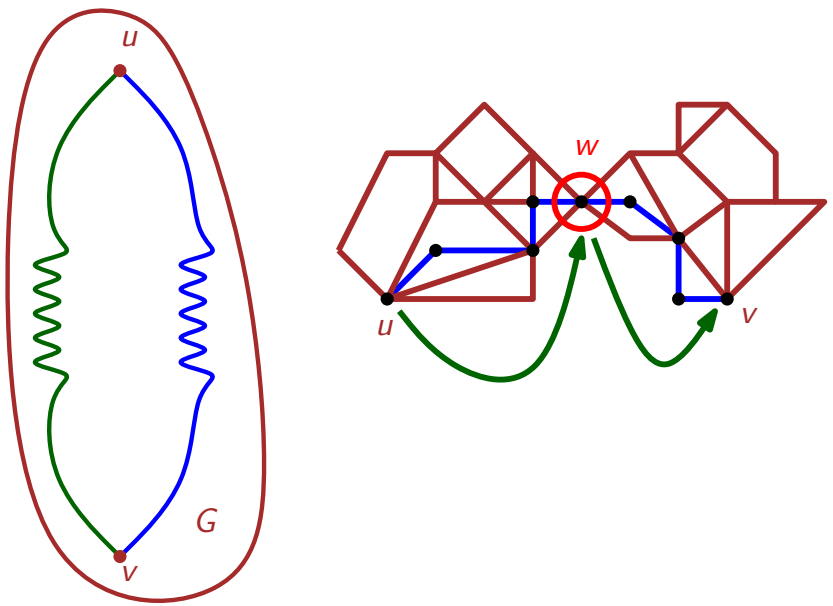
What is a biconnected or 2-connected graph? Is the following graph bi-connected?



# Basic Assumptions-II

What is a biconnected or 2-connected graph? Is the following graph bi-connected?




No! Above graph is not a bi-connected graph. Every path from $u$ to $v$ has to pass through $w$, so there can only be one path. $w$ is called cut-vertex.

# Preliminaries

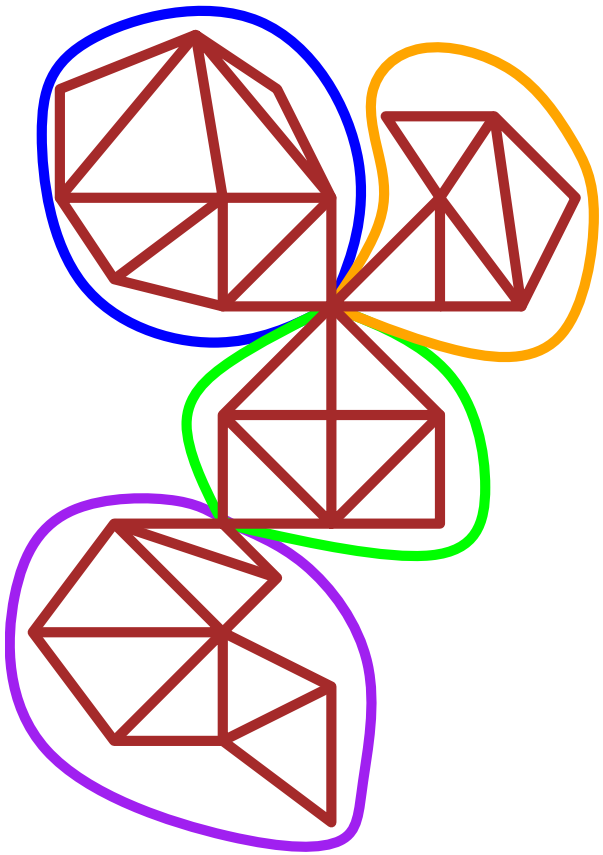


Figure: Biconnected Components

**Fact**

*Biconnected components can be tested for planarity separately.*

# Preliminaries

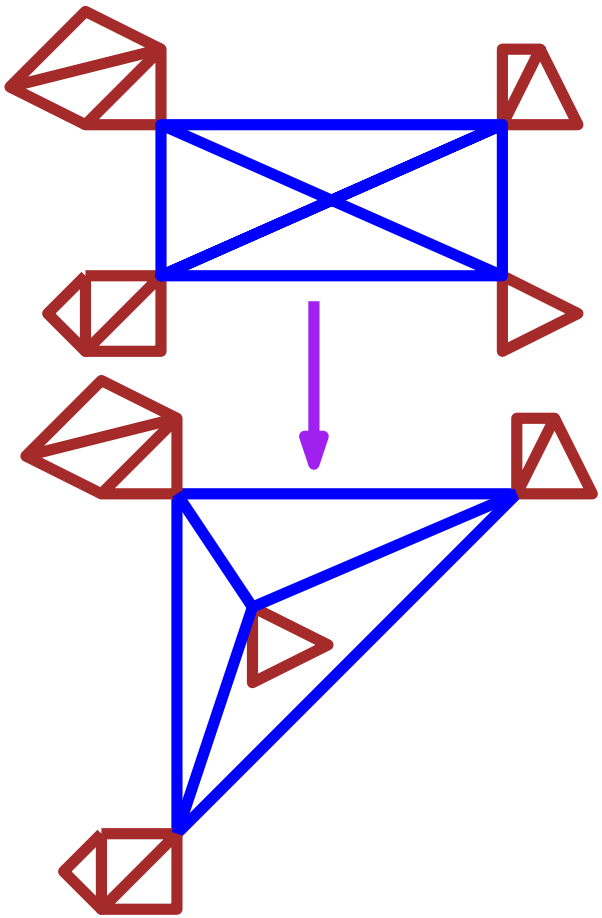


Figure: Biconnected Components

**Fact**

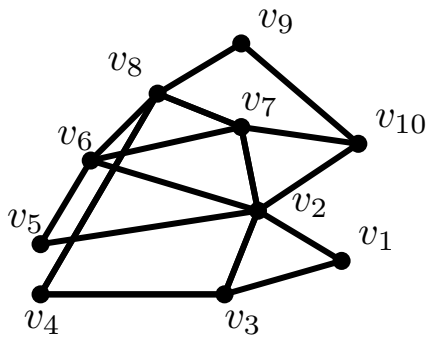
*Sometimes we may need to embed a biconnected component inside a face.*

# General Idea

Now Let us have a general idea of the algorithm.

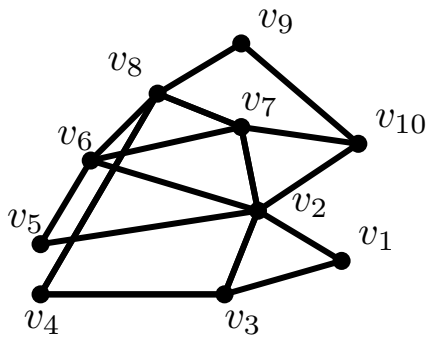
We will take a simple example. And see how we can test planarity following Hopcroft and Tarjan's Method.

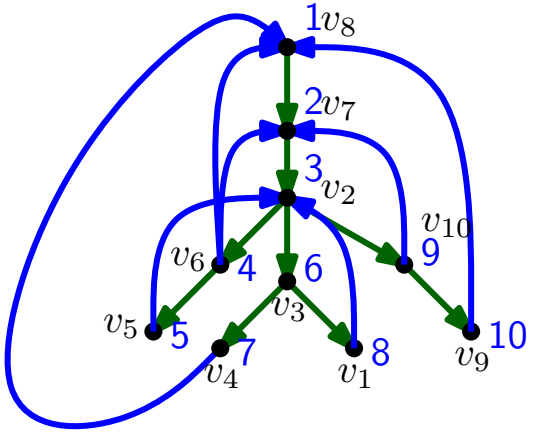
# General Idea




Idea: Given the graph above how will we test for planarity?

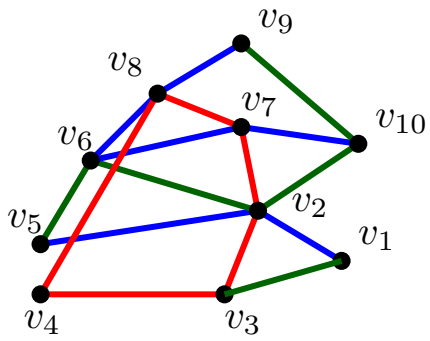
# General Idea

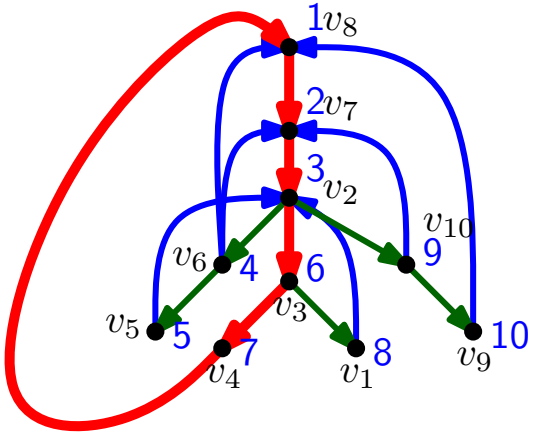



Idea: First we do a DFS, and mark the tree edges.

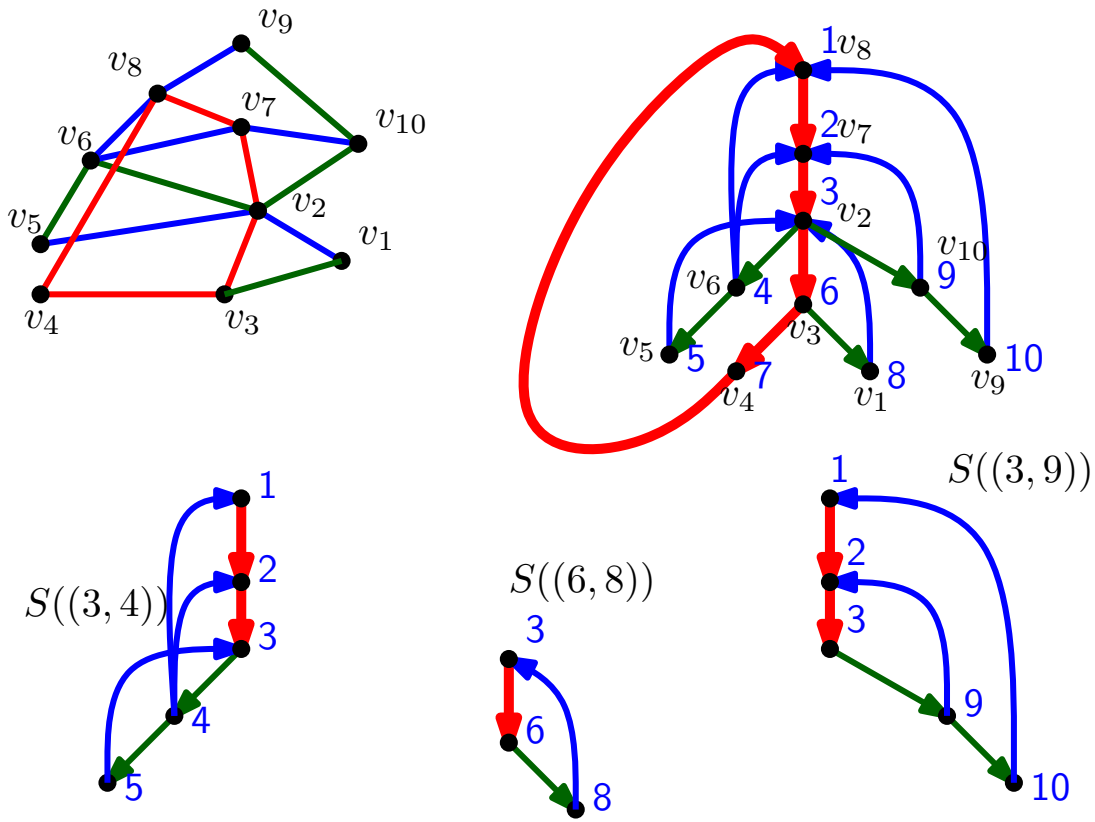
# General Idea



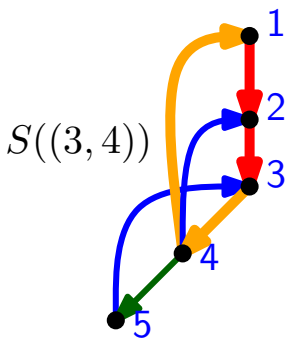

Idea: After suitably ordering the edges, we get a big cycle.

# General Idea



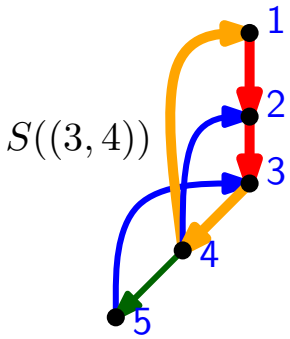

Idea: Emanating edges from the cycle form segments.

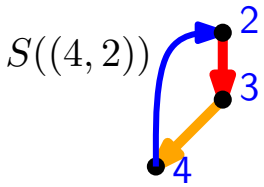
# General Idea

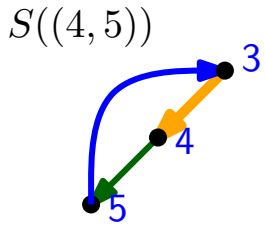



Idea: We test first segment for planarity.

# General Idea

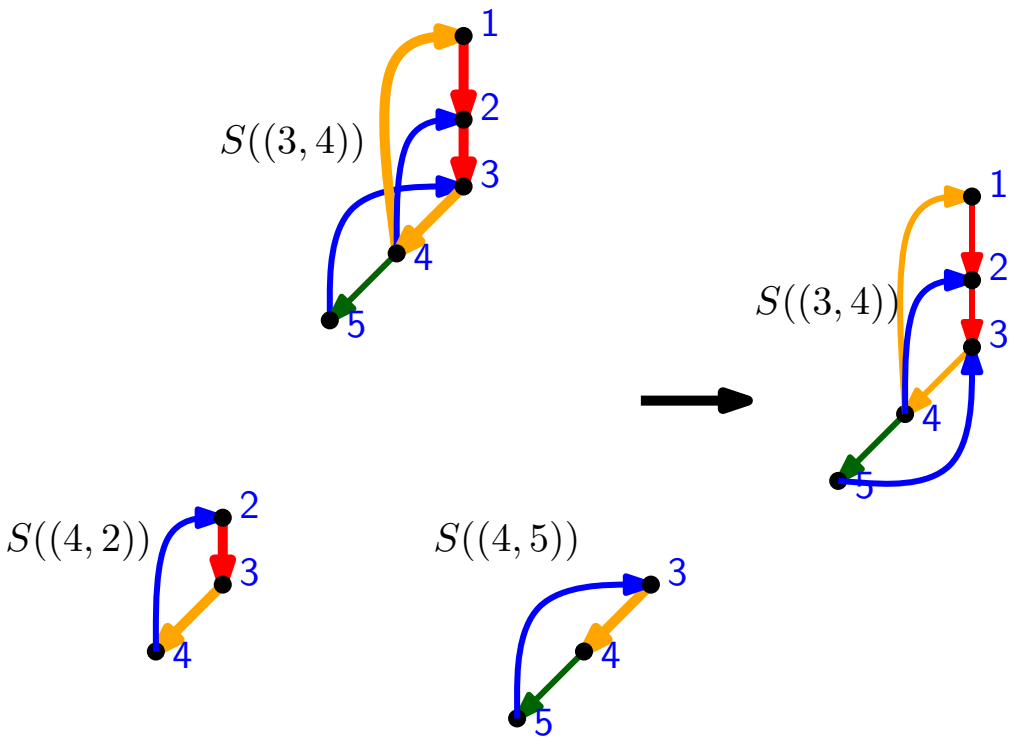


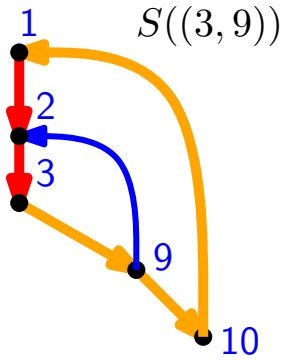

Idea: We compute the segments of the subgraph.

# General Idea



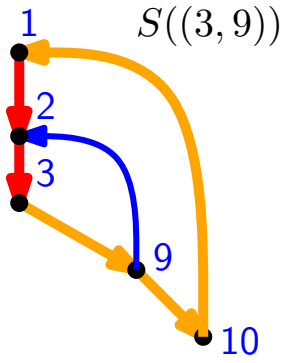

Idea: First segment is planar.

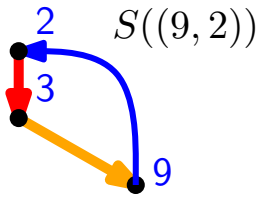
# General Idea




Idea: We test third segment for planarity.

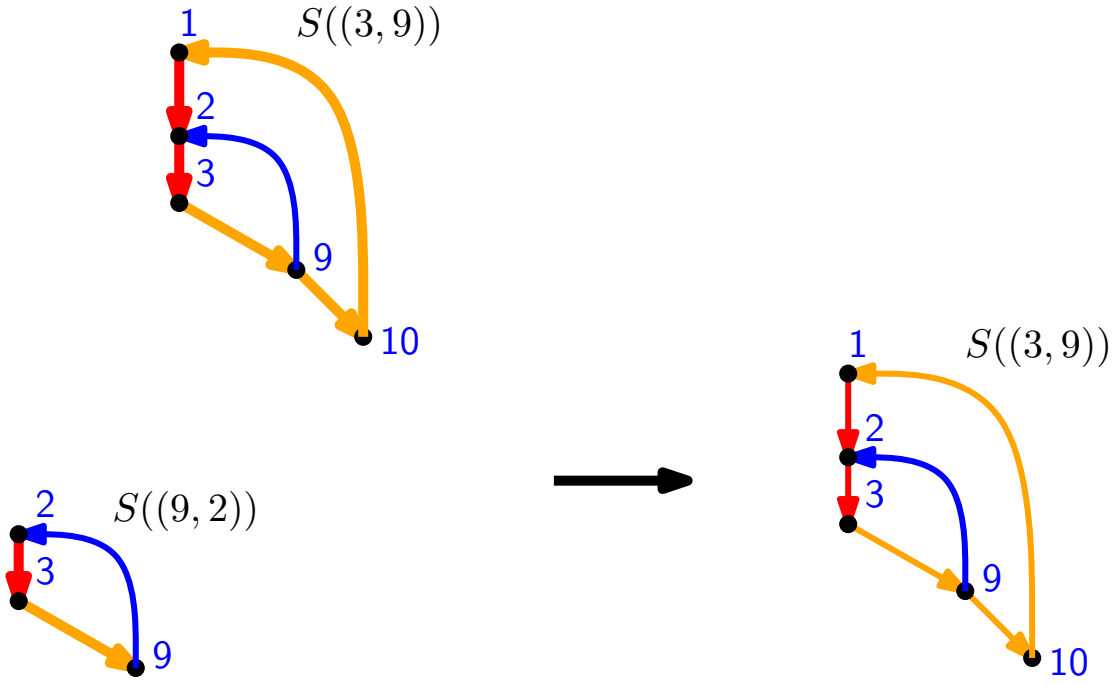
# General Idea



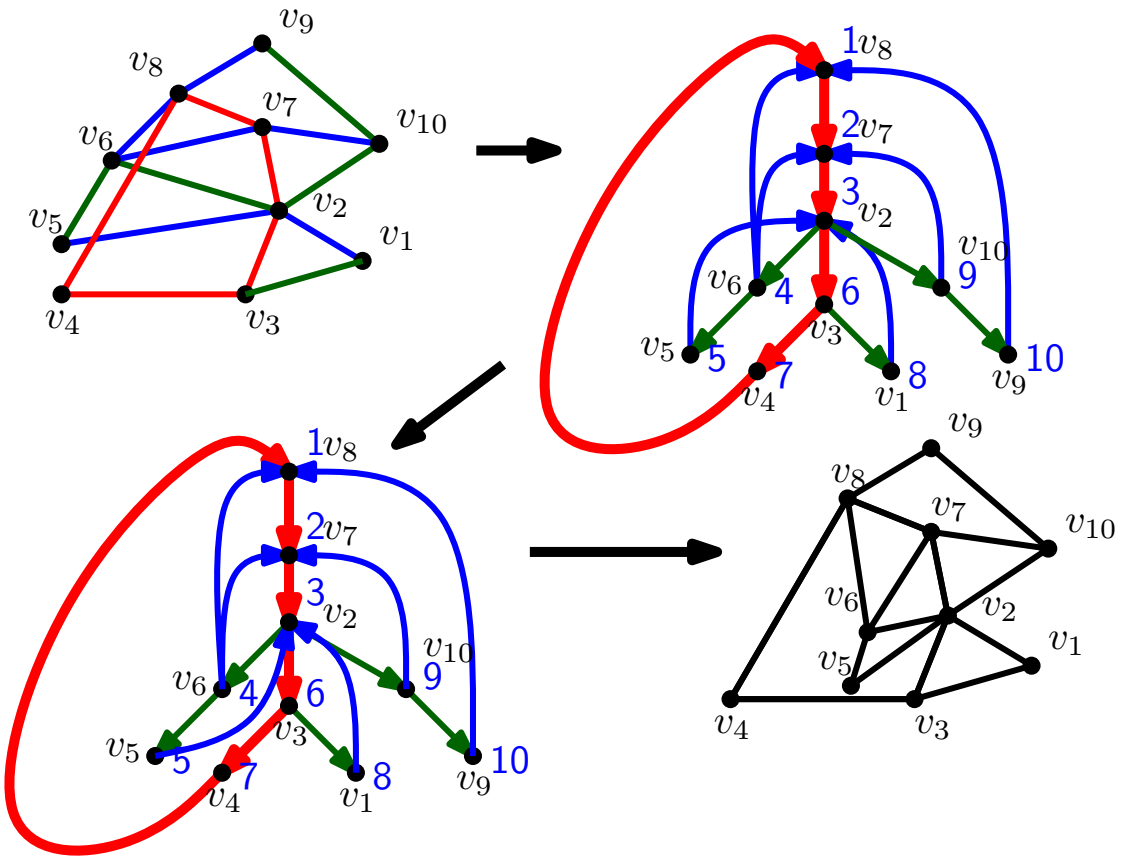

Idea: We compute the segments of the subgraph.

# General Idea



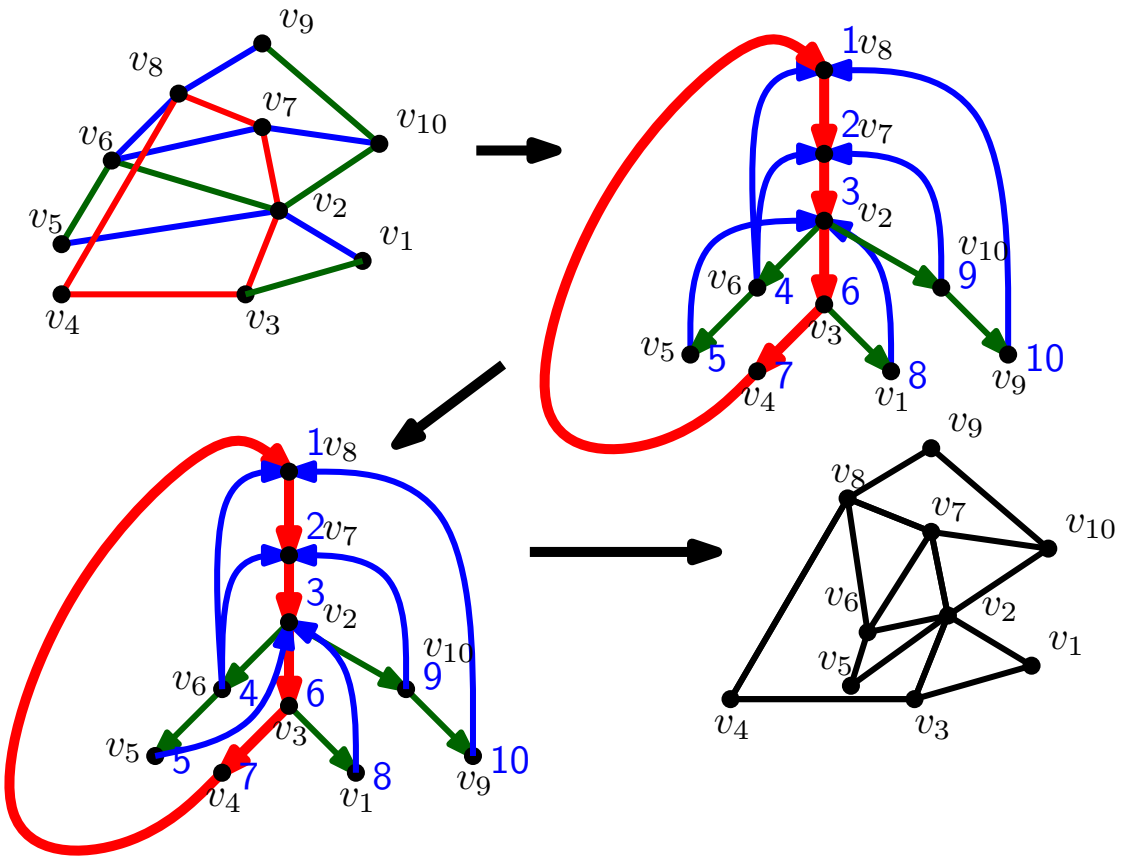

Idea: Third segment is planar.

# General Idea




Idea: We embed all the segments back in the original cycle.

# General Idea




Idea: Thus the above graph is planar.

# Planarity Testing Example : Summary

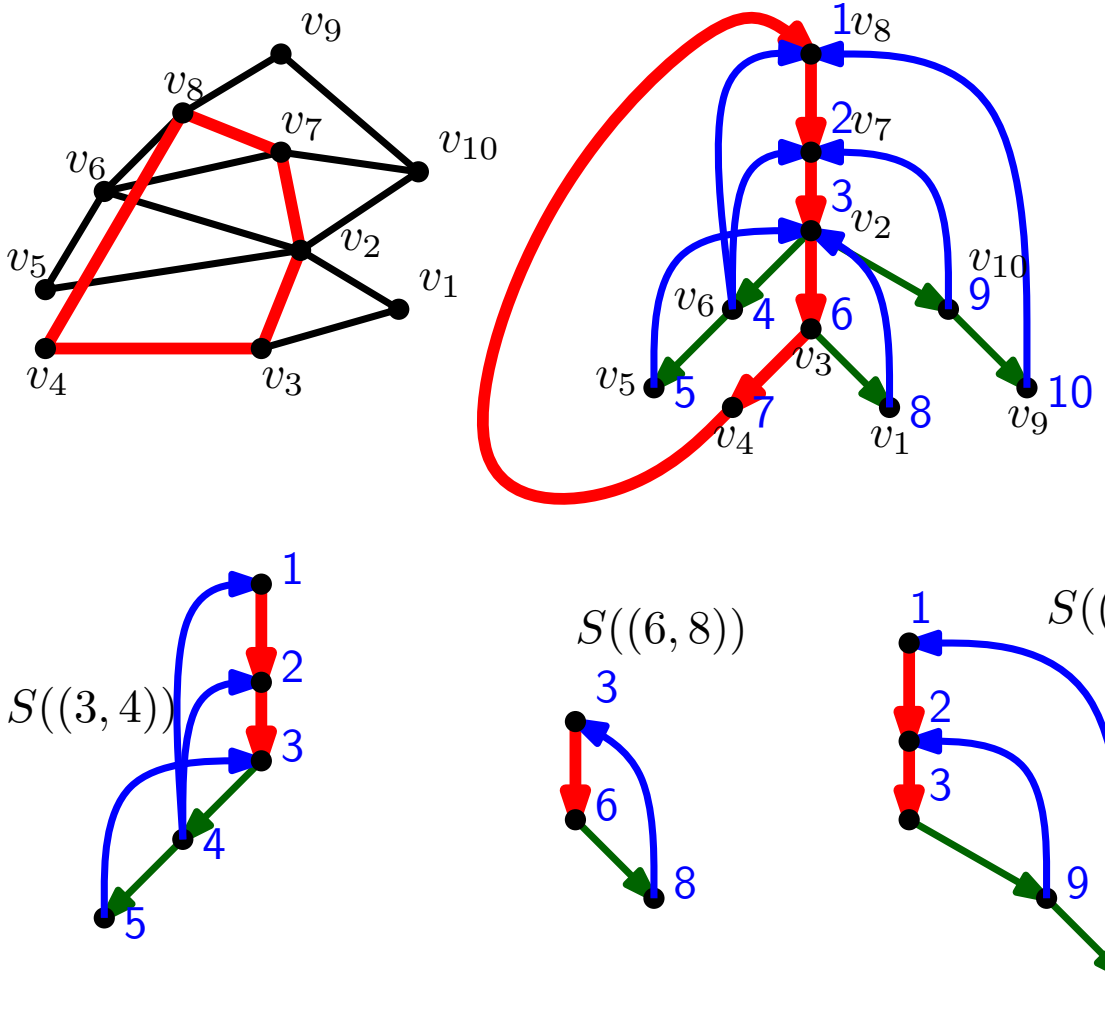



Figure: Testing of planarity of a simple graph

# Planarity Testing Example : Summary

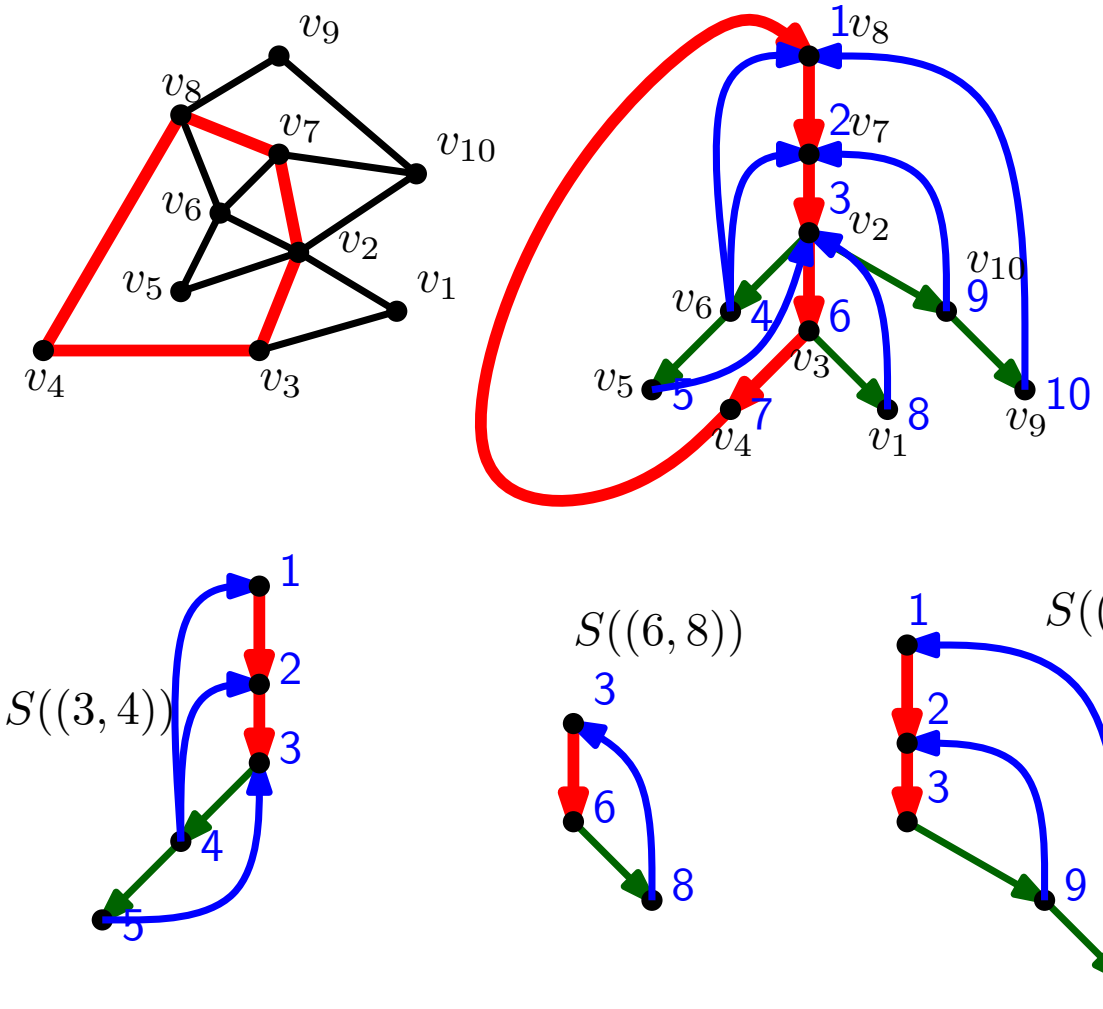



Figure: Testing of planarity of a simple graph

# Planarity Testing : Key Ideas

Data structures used are very simple, sets maintained as lists or double-ended lists at most.

But we do the recursive step in big-Oh of number of segments and lenght of spine.

There are four salient features of the algorithm.

# First Feature

First salient feature

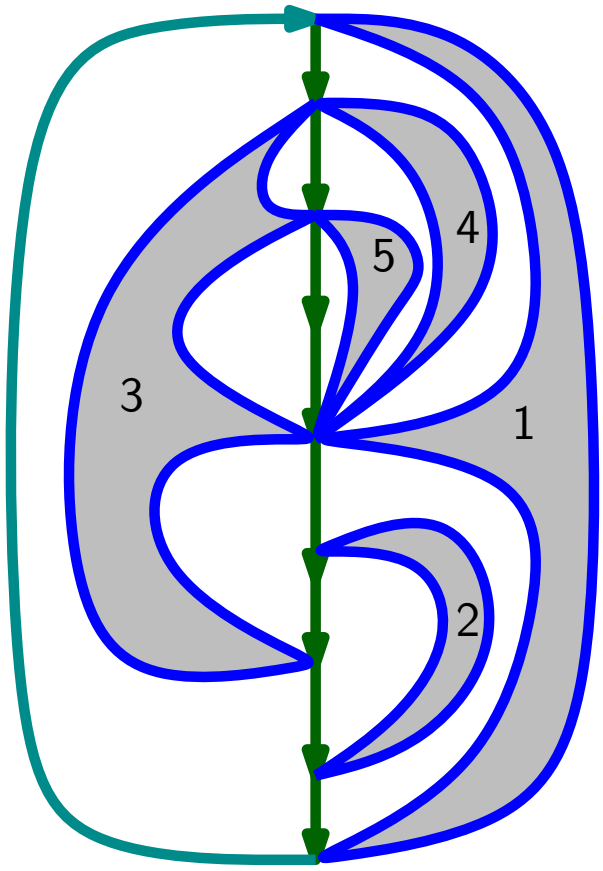



Figure: Outside-in Embed Order

The Edges are reordered in a certain way to embed segments outside in.

# Second Feature

Second salient feature

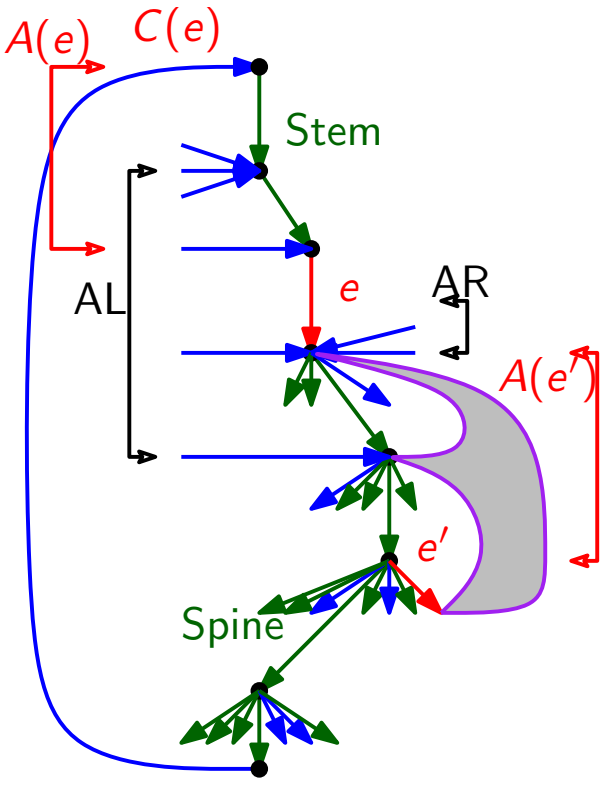



Figure: Bottom-up Segment Recursion

We find a primary cycle and embed segments recursively bottom-up on either side of the cycle.

# Third Feature

Third salient feature

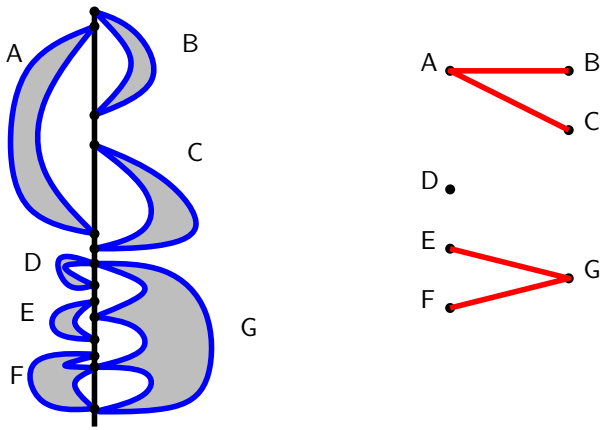



Figure: Interlacing graph

Interlacing graph is maintained to check conflicts between the segments.

# Fourth Feature

Fourth salient feature

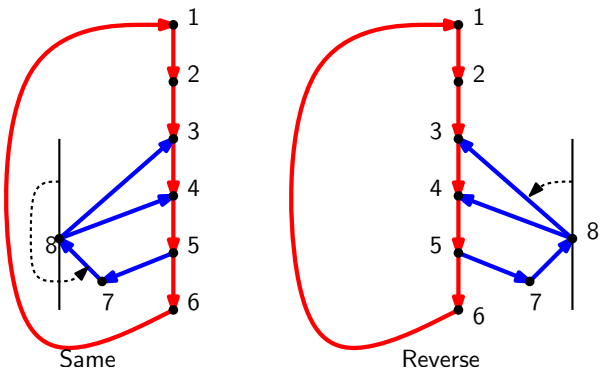



Figure: Left or Right Embedding

To embed we need to keep blocks of conflicting segments together, which allows us to switch side efficiently, as well as telling us whether to order adjacency lists clockwise or anticlockwise.

# Planarity Testing : Key Ideas

Four salient features of the Hopcroft and Tarjan's Algorithm.

- The Edges are reordered in a certain way to embed segments outside in.
- We find a primary cycle and embed segments recursively bottom-up on either side of the cycle.
- Interlacing graph is maintained to check conflicts between the segments.
- To embed we need to keep blocks of conflicting segments together, which allows us to switch side efficiently, as well as telling us whether to order adjacency lists clockwise or anticlockwise.

# Planarity Testing : Key Ideas

Four salient features of the Hopcroft and Tarjan's Algorithm.

- The Edges are reordered in a certain way to embed segments outside in.
- We find a primary cycle and embed segments recursively bottom-up on either side of the cycle.
- Interlacing graph is maintained to check conflicts between the segments.
- To embed we need to keep blocks of conflicting segments together, which allows us to switch side efficiently, as well as telling us whether to order adjacency lists clockwise or anticlockwise.

We have already used first three ideas in the example shown before.

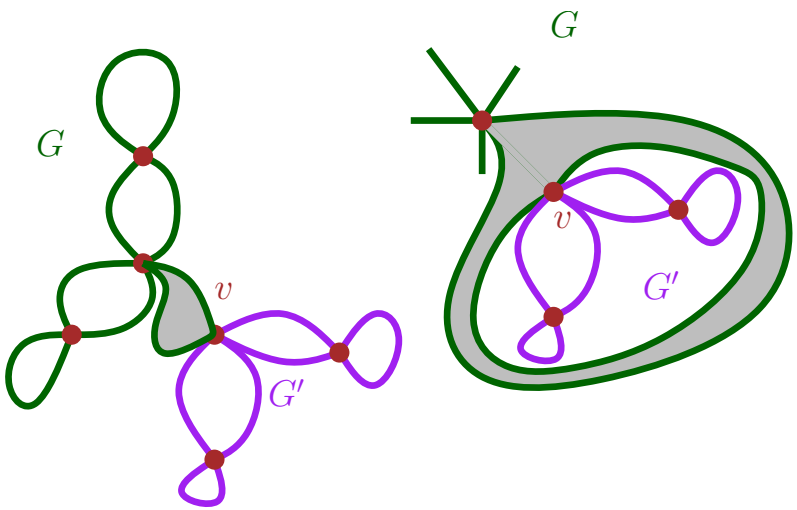
# Basic Assumptions-I

We assume the following.

- The given graph $G$ does not have self loops or multi-edges.
- $G$ is undirected and satisfies $m \leq 3n - 6$.
- $G$ is connected. This also means there won't be isolated vertices.

# Basic Assumptions-II

We further assume.




- $G$ is bi-connected[2]. If $G$ is not, we find cut-vertices and test the planarity of each bi-connected component separately.
- We can embed the component in a face incident to the cut vertex[3].

[2] So no vertex is of degree 1.

[3] Actually we can make any face of a planar graph the outer face.

# Preliminaries

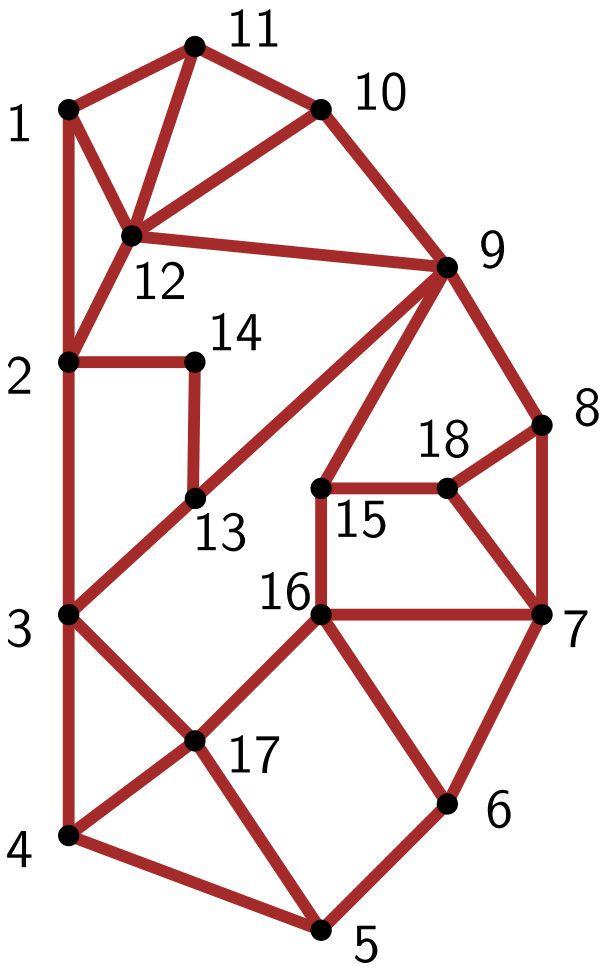
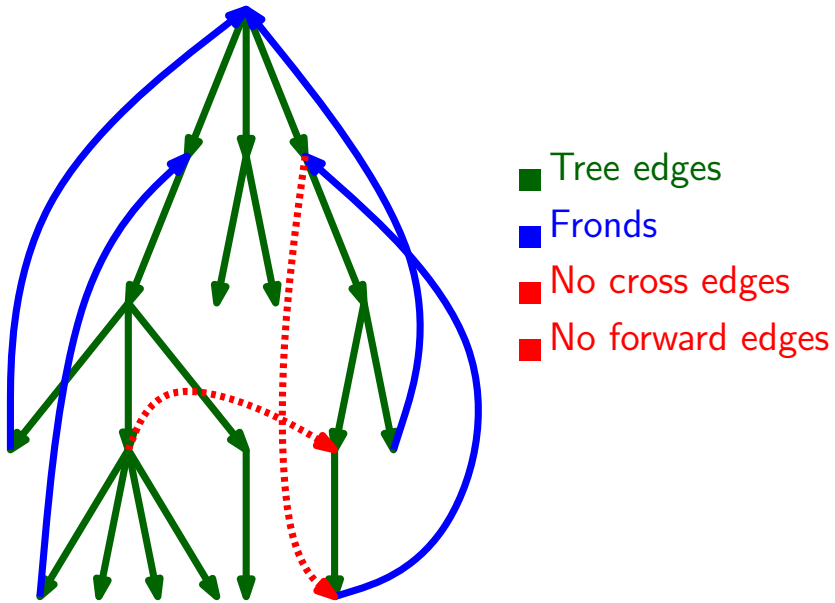



Figure: DFS numbering

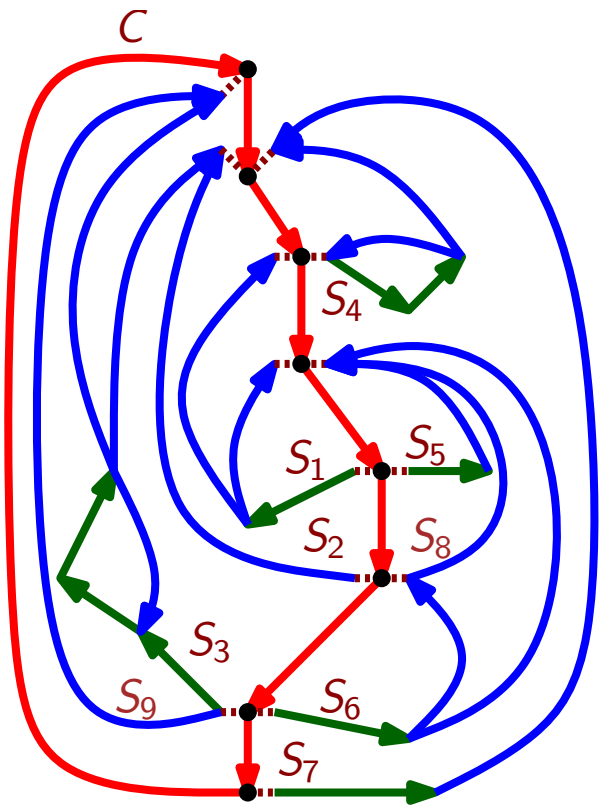
We rename each vertex as its dfs number for the sake of clarity.

# Preliminaries



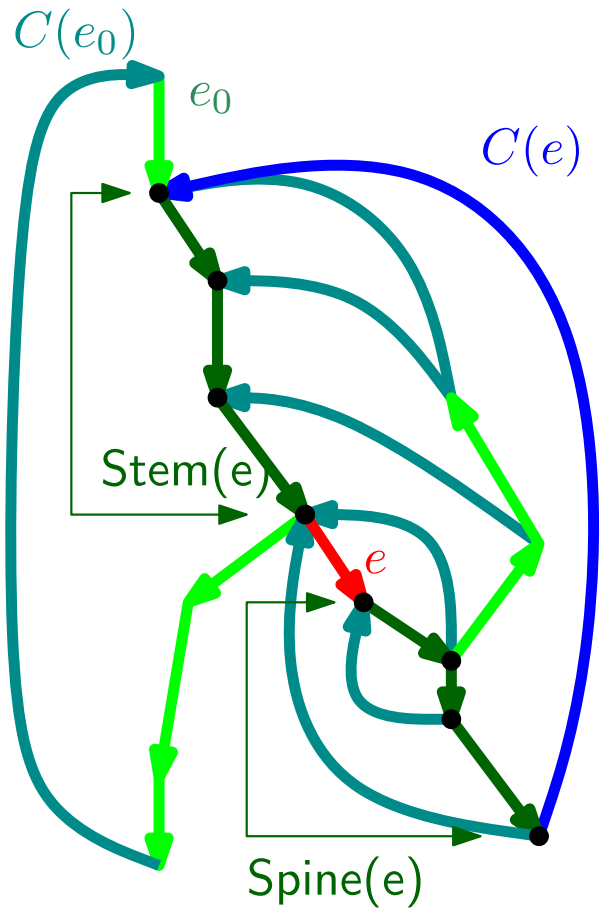

In DFS we also mark the edges as tree edges or back edges(fronds). There are no cross edges or forward edges.

# Preliminaries




Every cycle is simple cycle in planar embedding.
A cycle divides the graph into segments to be embedded.
Each tree-edge and frond emanating from the cycle is a separate segment.
$C$ exists because of biconnectivity.

# Preliminaries

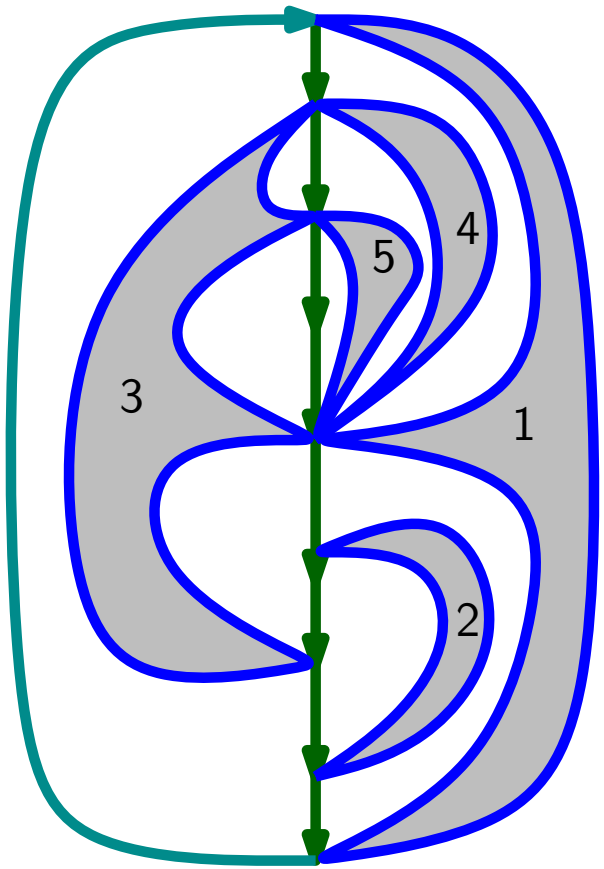



Segment of $e$, $S(e)$

We embed the segments recursively. Each segment (also called branch) corresponds to a tree-edge or frond emanating from the cycle in question. We define *stem* and *spine* of edge $e$ which forms cycle $C(e)$ with $e$ and a frond.

The segment can be embedded on one side of stem but on both sides of spine.

# Preliminaries




Embedding order

We wish to embed the segments outside in, so any conflicts can be easily detected.
For this we take the segments bottom up and in a particular sorted order for the same node.

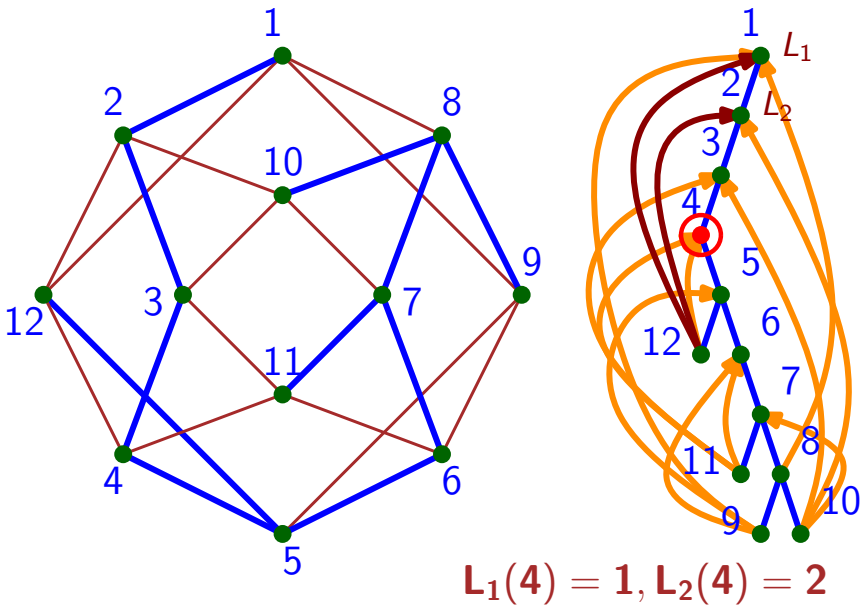
# Outline of Algorithm - Stage I

We are now in a position to present our algorithm.

PREPROCESSING STEP - 1

- First we find out bi-connected components of the given graph $G$.
- Test planarity of each bi-connected component individually. If all are planar then $G$ is planar.
- We assume for clarity that bi-connected component itself is $G$.
- We do a depth-first-search and compute all tree edges and backedges (fronds).
- We rename vertices with their $dfs$ numbers.

# Oultline of Algorithm - Stage II




We need to sort the adjacency list of each node. So we define lowpoints — $L_1$ and $L_2$ for each vertex.

# Lowpoints — $L_1$ and $L_2$

We define lowpoints $L_1(u)$ and $L_2(u)$ for each vertex as follows. Let $D(u)$ be set of all descendants of $u$ and $T(u)$ be set of all neighbours of $D(u)$.

**Definition**

$L_1(u) = min\{T(u)\}$
$L_2(u) = min\{T(u) - \{L_1(u)\}\}$

# Lowpoints Facts — $L_1$ and $L_2$

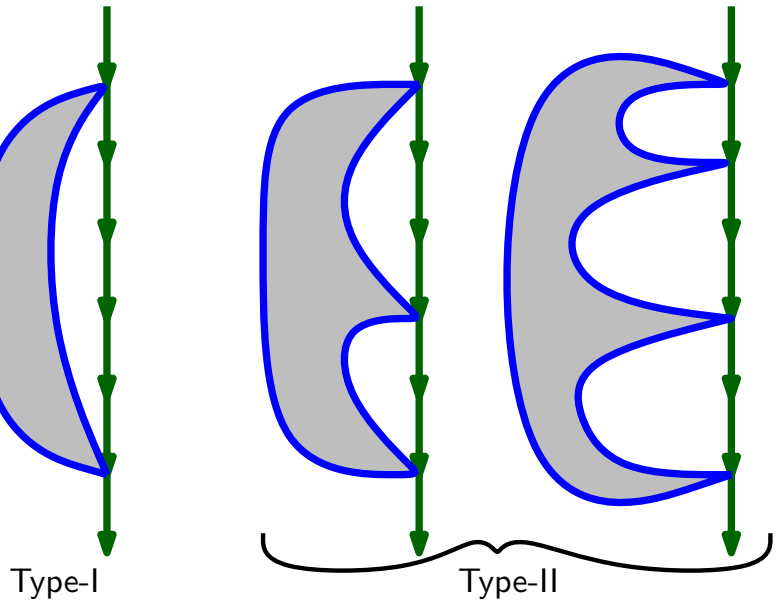
**Fact**

*Since $G$ is 2-connected graph, it has no cut-vertices, so we can conclude that $L_1(v) < u$ whenever $u$ is parent of $v$ and $u$ is not the root node. It follows that $L_2(v) \leq u$.*

**Fact**

*Low-points are well-defined, since every vertex has atleast 2 neighbours.*

# Oultline of Algorithm - Stage II

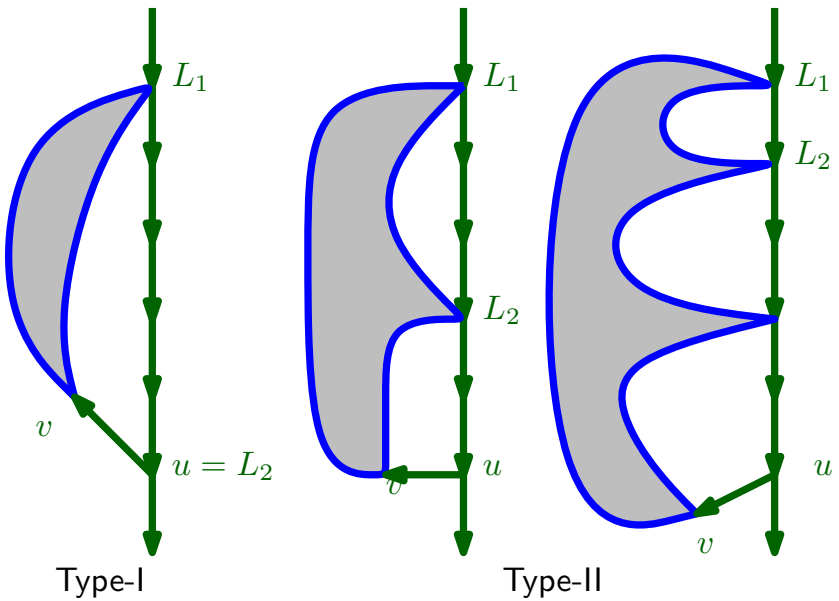



Type I and Type II

There are two types of segments. One with only two attachments is called Type I and another with multiple attachments is called Type II.

# Oultline of Algorithm - Stage II

How to calculate Type-I and Type-II segments.




Type I and Type II

A branch (segment) $B_u(v)$, where $u$ is parent of $v$, with $L_2(v) = u$ is a type I branch. If $L_2(v) < u$ it is a type II branch.

# Oultline of Algorithm - Stage II

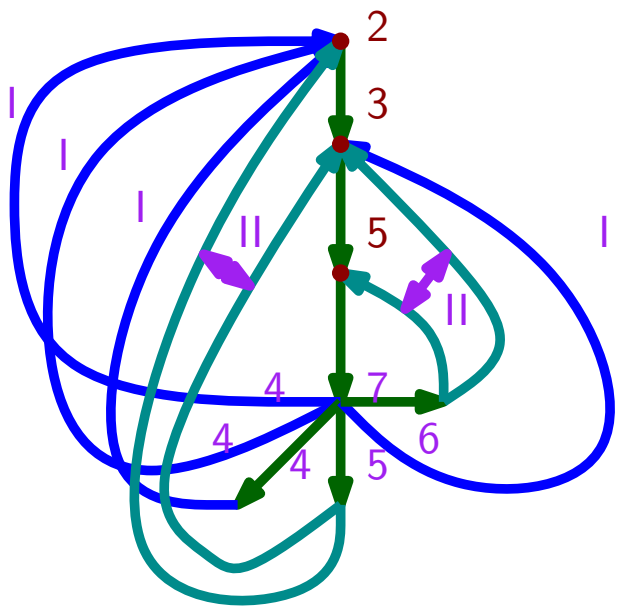
**Fact**

*Any number of Type-I with common attachments can be embedded inside each other. Once we get Type-II, only one can be embedded.*

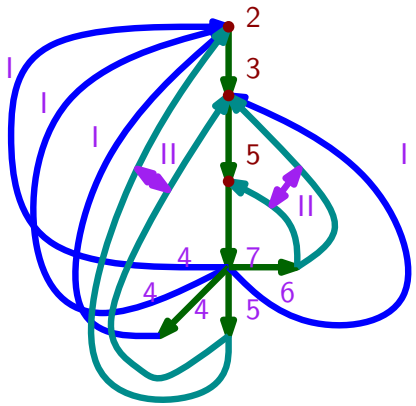
So we need to sort the adjacency list so that Type-I segments occur before Type-II.
We reorder edges of a vertex so that components are added on the tree in increasing depth of where they are attached on the DFS tree.

# Oultline of Algorithm - Stage II

Some segments with same starting point are shown below with their sorting keys.



# Oultline of Algorithm - Stage II




The weight of edge $uv$ is defined as

$$wt_u(v) = \begin{cases} 2v & \text{if } uv \text{ is a frond with } v < u \\ 2L_1(v) & \text{if } u \text{ is parent of } v \text{ and } L_2(v) = u \\ 2L_1(v) + 1 & \text{if } u \text{ is parent of } v \text{ and } L_2(v) < u \\ 2n + 1 & \text{otherwise} \end{cases}$$

# Reordering of Edges — Example

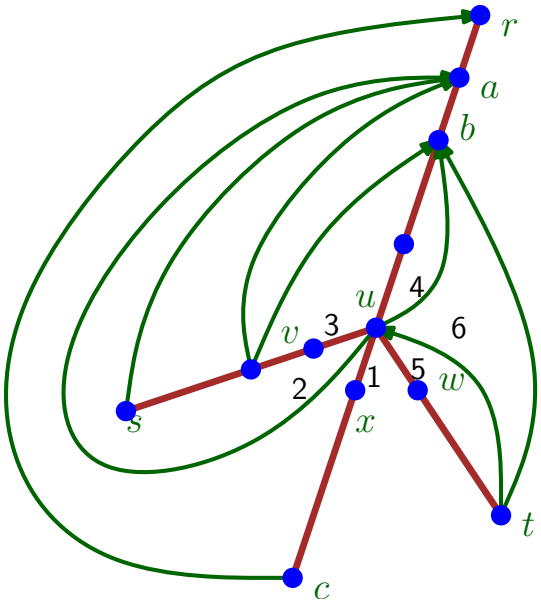



Figure: DFS tree and Lowpoints

At vertex $u$ the branches (segments) are embedded in the order $B_u(x)$, $ua$, $B_u(v)$, $ub$, $B_u(w)$, and $ut$

Since $wt_u[x] = 2r < wt_u[a] = 2a < wt_u[v] = 2a + 1 < wt_u[b] = 2b = wt_u[w] < wt_u[t] = 2n + 1$.

# Outline of Algorithm - Stage II

## Preprocessing Step - 2

- During DFS we also calculate two lowpoint arrays $L_1(v)$'s and $L_2(v)$'s.
- We sort the adjacency lists according to the criteria of lowpoints. Use bucket sort to do it in overall linear time (keys range from 1 to $2n + 1$).
- There is one cycle starting from root ending there, and we try to embed $G$ branch by branch recursively.

# Outline of Algorithm - Stage III

**Theorem**

*$G$ is planar iff each branch (segment) can be embedded on one side of its stem (defined as strongly planar). Stem is the part from $L_1$ point to the beginning of the branch.*

This enables us to check conflict only on the basis of attachments to the cycle.

# Outline of Algorithm - Stage III

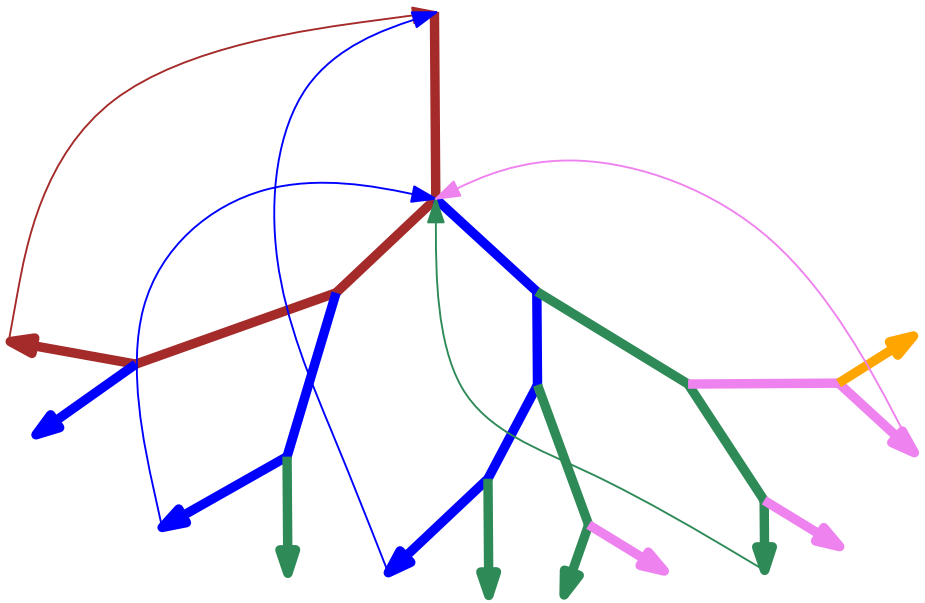


Figure: Cycle of segments

The way ordering of edges is done, the cycle is always formed by the first of children till a frond is encountered to the stem.

Further root has one child only (because of biconnectivity).

# Outline of Algorithm - Stage III

Ordering of $Adj[u]$ by weights is not sufficient to guarantee that an embedding is possible without further refinement.

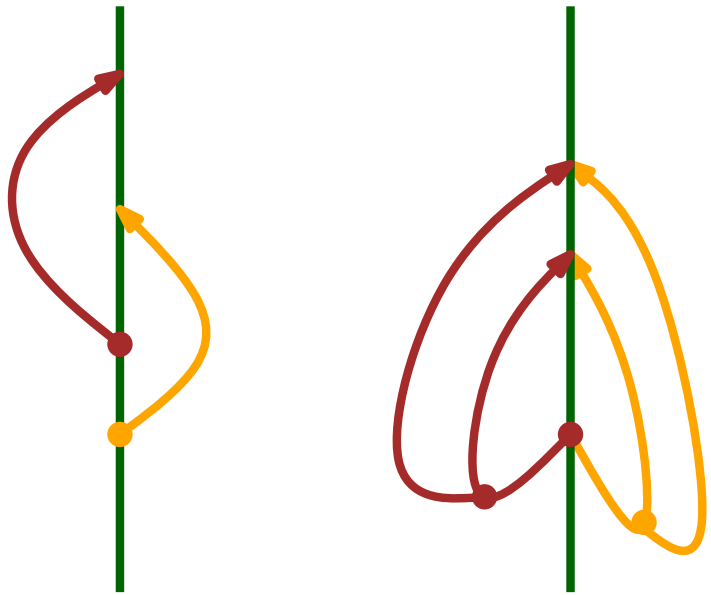
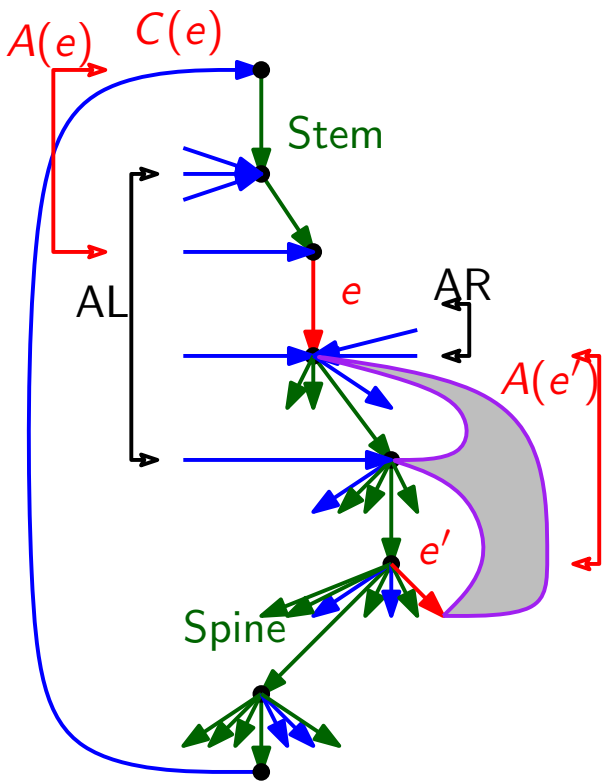


Figure: Conflicting Fronds and Segments

If there is a conflict then we must embed on the opposite sides of the current path from root.

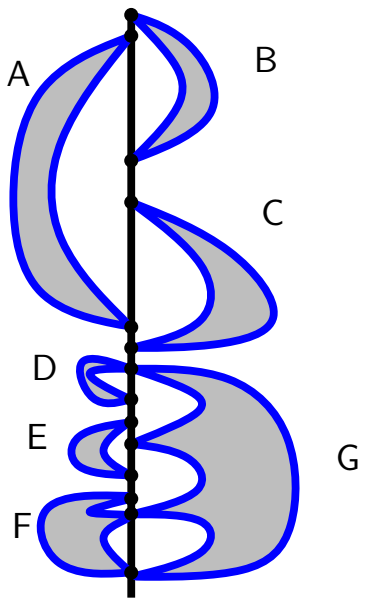
# Outline of Algorithm - Stage III

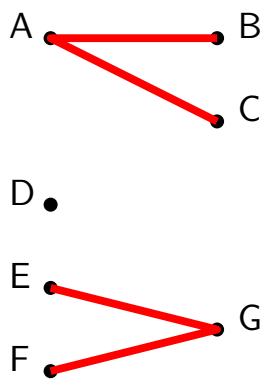



We process segments bottom-up.
We maintain two lists of left and right attachments $AL$ and $AR$ as stacks.
These are enough to check if there will be conflicts.
But sometimes we need to move the segments from left to right!!!
We need to return $AL$, i.e., $A(e)$, after successful checking of planarity of $S(e)$.

# Outline of Algorithm - Stage III

For keeping tabs on which segment conflicts which one, we compute interlacing graph.




Interlacing graph must be bipartite. While switching we switch all the segments in a connected component.

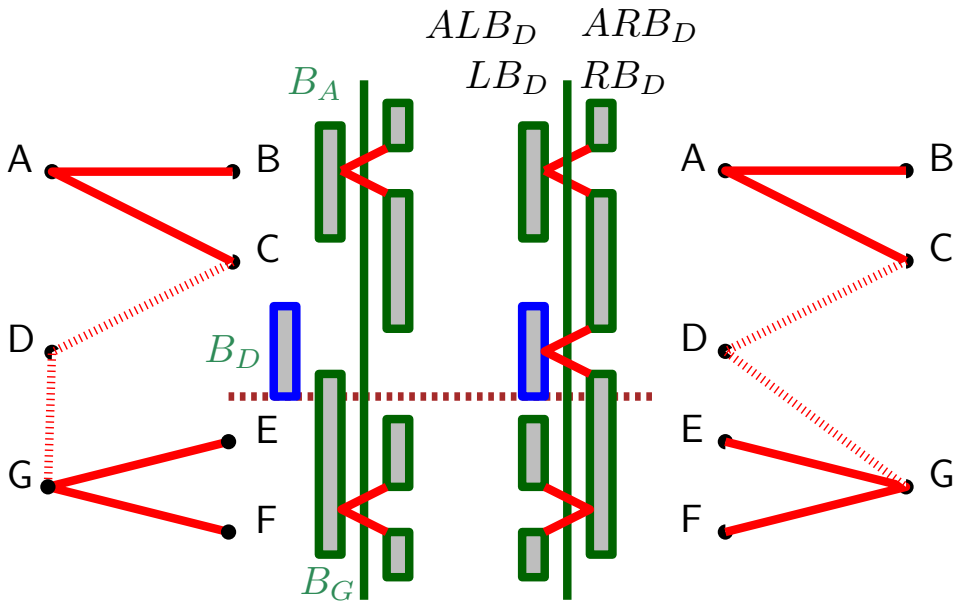
# Outline of Algorithm - Stage III

It is easy to see that we have now an $O(n^3)$ algorithm. Computing interlacing graph itself is $O(n^2)$ in each recursive step and there are $O(n)$ recursive steps.

To do it in overall $O(n)$ we need to compute interlacing graph implicitly, without computing all its edges. It sufficies if connected components remain connected.

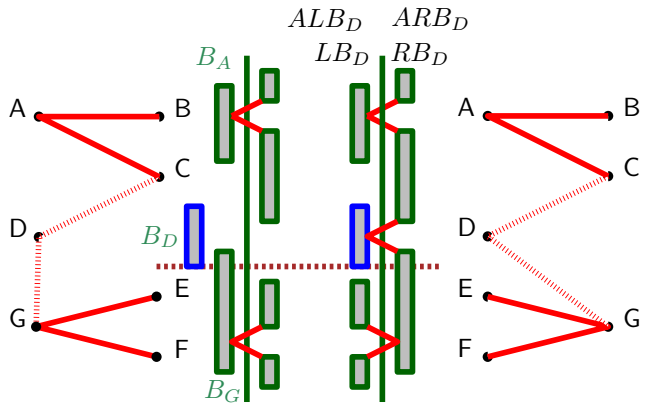
Even $O(n)$ time per recursion won't do. We need to do it in linear time of number of segments and length of spine.

# Outline of Algorithm - Stage III




For that we need to treat all the conflicting segments as a block. When a block is known to conflict with another block, we combine them (union) and stitch the attachment lists (they are two $ALB$ and $ARB$ for each block).

# Outline of Algorithm - Stage III

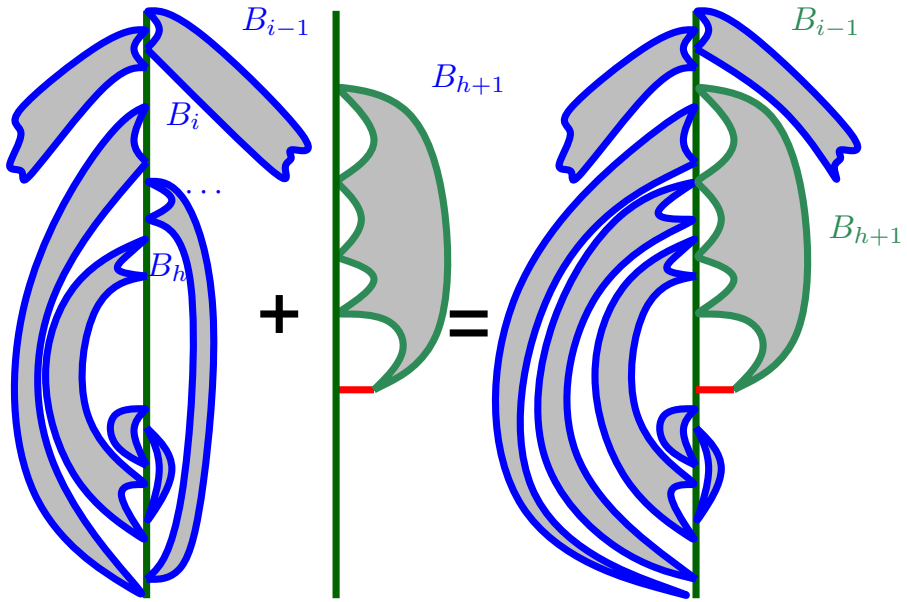



Now we do not switch side for individual segments, we switch a block. Switching involves linking it to other side ($LB$ and $RB$ for each block). Since all blocks are stitched together, all of them switch sides.

For embedding we assign the side a segment ,i.e., left or right after the whole process is over.

# Outline of Algorithm - Stage III

Implicit computation of interlacing graph.




Maintenance of Interlacing graph over the tree-edge (current segment) is easy. There is an order to blocks (can be seen as stack).

# Outline of Algorithm - Stage III

We are in a position to give our main processing step.

We embed the whole graph using the DFS tree $T$ branch by branch bottom-up. First cycle will have empty stem.

We first determine an ordering of $adj(u)$ for each $u$ so that the segments are guaranteed to permit an embedding when $G$ is planar.

# Outline of Algorithm - Stage III

## Main Processing Step

Algorithm to check strong planarity of $S(e)$

- Since $G$ is bi-connected, initially we will have a cycle $C(e)$ found by following the first children.
- So we have an embedding of $C(e)$ in the plane, with the segments to be arranged around $C(e)$.
- Each segment in turn is embedded recursively bottom-up. We get its list of attachments on the stem.
- Then, following Hopcroft and Tarjan, we keep two linked lists of fronds, $AL$ and $AR$ containing the fronds and branches embedded on the left of $C(e)$ and on the right of $C(e)$ , respectively.
- $AL$ and $AR$ can be viewed as stacks, whose tops contain the number of the vertex currently marking the upper limit in the tree to which fronds may be embedded.

# Outline of Algorithm - Stage III

## Main Processing Step (contd.)

- We also construct the conflict graph of branches in $LB$ and $RB$ for each block, as $IG$ which is a bipartite graph.
- Then $AL$ and $AR$ are unions of $ALB$'s and $ARB$'s of each block.
- After embedding every segment we return $A(e)$, the attachments on the stem (i.e. by default $AL$).
- We also assign the side of each segment before returning.

# $K_5$ Planarity Testing

How does $K_5$ fare in our algorithm?

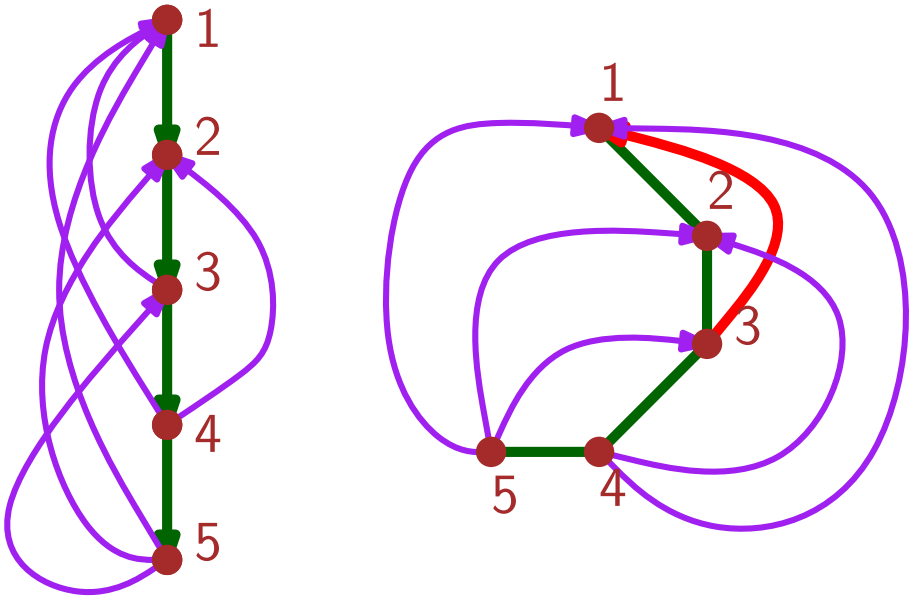



Figure: Testing of Planarity for $K_5$

Last segment conflicts with both $AL$ and $AR$. (Only 1-level recursion.)

# $K_{3,3}$ Planarity Testing

How does $K_{3,3}$ fare in our algorithm?

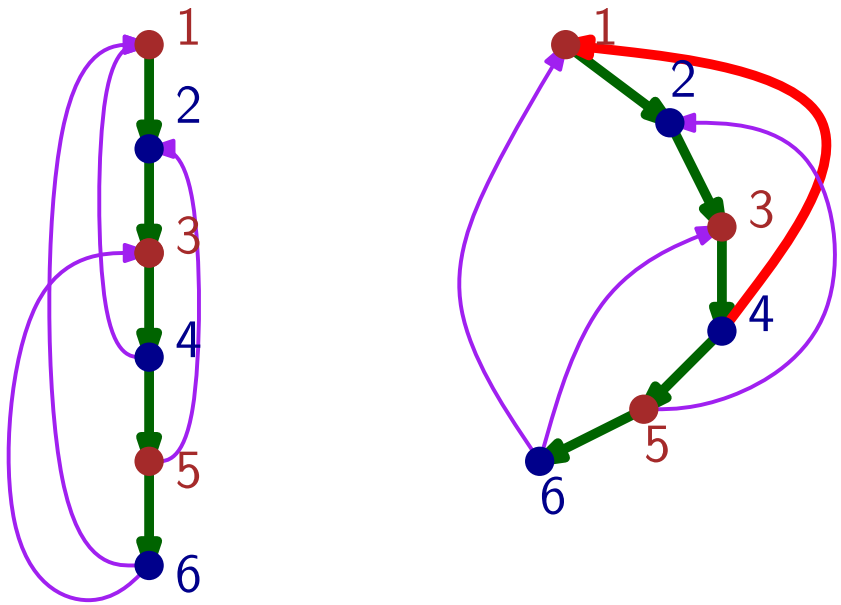



Figure: Testing of Planarity for $K_{3,3}$

Last segment conflicts with both $AL$ and $AR$. (Only 1-level recursion.)

# Planar Embedding of Graphs

# Embedding Phase of Planarity Algorithm

Though Hopcroft and Tarjan gave the algorithm to test the planarity, they could not use it to give a planar embedding[5].

It took Mehlhorn and Mutzel another twelve years to show how Hopcroft and Tarjan's algorithm can be used to embed a planar graph[8].

It simply means that embedding is not as obvious as it seems.

# Embedding Phase of Planarity Algorithm

**Theorem**

*Let $G$ be a biconnected planar graph. Then we can embed $G$ in the plane in linear time[5, 8, 7].*

# Embedding : Key Ideas

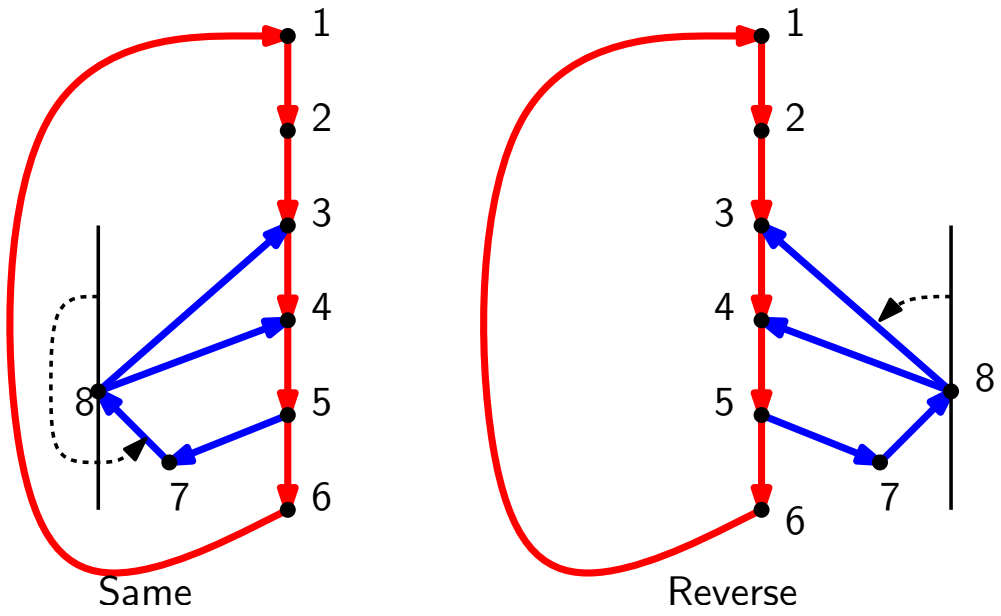



Figure: Adjacency List: Clockwise or Anti-clockwise?

Adjacency lists are arranged clockwise or anti-clockwise depending on whether the segment is embedded left of the cycle or right of the cycle.

# Embedding : Key Ideas

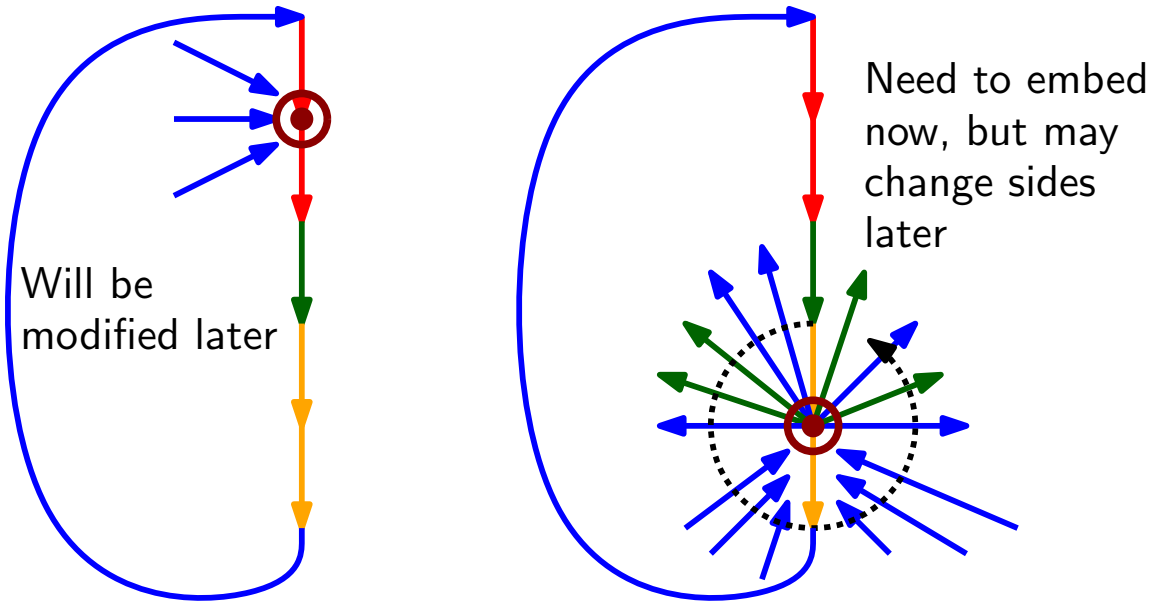



Figure: Adjacency List: Finalising at spine

At the spine we need to order the vertices on both sides. The players are outgoing fronds, outgoing tree edges, cycle edges, and incoming fronds.

# Other Embeddings

The embedding that we have obtained has curved edges.
Sometimes we need different criteria for embedding, such as:

- Edges might be needed to be straight.
- Further, vertices might be needed on the grid.
- Furthermore, area might be needed to be minimized.
- Or we need edges made of orthogonal segments (then additionally vertices will need to be rectangular regions, for degree $> 4$).

# Straight Line embedding

Straight Line Embeddings, what are they?

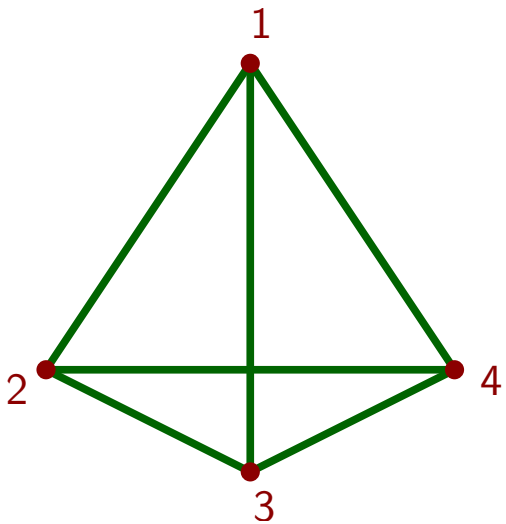



Figure: Straight Line Embedding of $K_4$

Question is: Given a planar graph $G$ does straight line embeddings always exist? How to compute it?

# Straight Line embedding

Straight Line Embeddings, what are they?

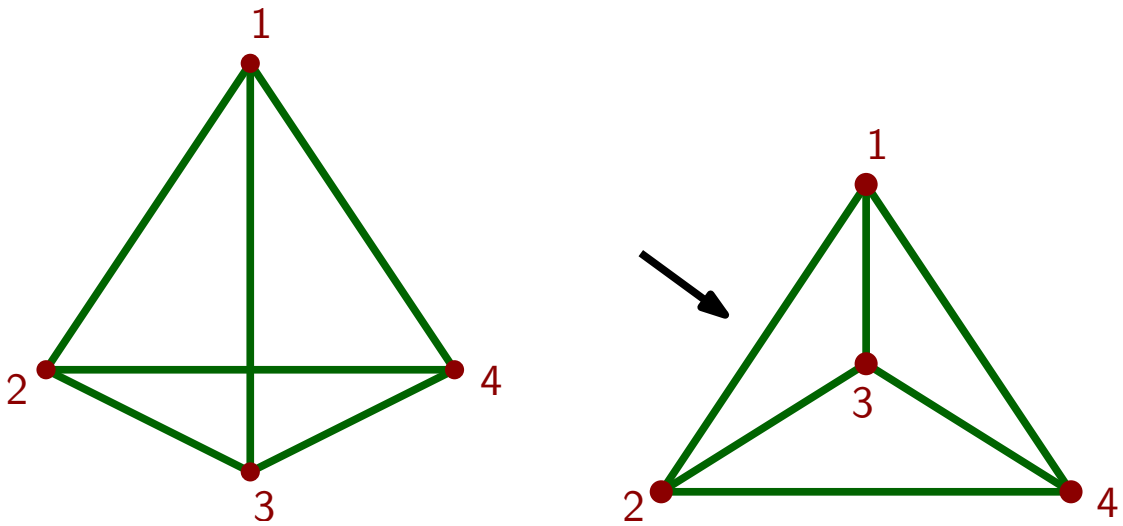



Figure: Straight Line Embedding of $K_4$

Question is: Given a planar graph $G$ does straight line embeddings always exist? How to compute it?

Answer is: Yes! and we give an idea of how next.

# Straight Line embedding: Basic Idea

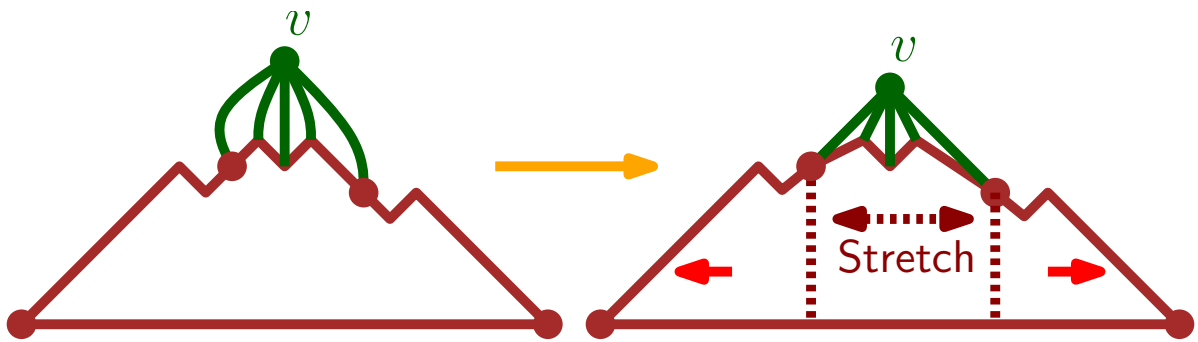



Figure: Straight Line Embedding

First we triangulate the planar graph. Start with an outer edge then add vertex after vertex.

We choose v such that it is connected to all consecutive nodes on the chain.

Then we shift the other vertices on the chain either right or left. How much? Refer [7, 1, 9].

# Straight Line Planar Embedding Theorem

**Theorem**

*Any planar graph with n nodes has a straight line embedding into the $2n-4$ by $n-2$ grid such that edges are mapped into straight-line segments. Also such an embedding can be constructed in $O(n \log n)$ time[7, 1, 9].*

# Tutte's Theorem

**Theorem (Tutte's)**

*If $G$ is a 3-connected planar graph, then $G$ has a convex embedding in the plane.*

This is a stronger result that Kuratowski's theorem.

# Conclusive Remarks

# Open Problems

Several research openings still now.

- Planar graphs which admit straight-line grid drawings on grids of linear area.
- Planar graphs which admit unit length straight-line edges.
- Planar graphs can be generalised to higher dimension where we have hyper-faces.
- If we allow crossings, then sometimes it makes sense to minimise crossings.
- There is a concept of book thickness of graphs. We embed the graph such that vertices are in spine and edges can be in the pages. We have to minimise number of pages.

# Summary

In conclusion

- We studied the concept of graph planarity.
- We gave a short proof of Kuratowski's theorem.
- Next we saw how we can answer queries about planarity of graph.
- We looked into the planar embeddings.
- Now we are for the final concluding remarks.

# Also read

You may also read

- Harary's book for basic graph theory[3].
- Hopcroft and Tarjan's paper[5] for linear planarity testing.
- Mehlhorn and Mutzel's paper[8] for linear planar embeddings.
- S. Even, A. Lempel, and I. Cederbaum's work[2] for a simpler embedding algorithm.
- Chapter 4 of Mehlhorn's book[7] for further details

# Acknowledgments

Thanks to Dr Subir Ghosh, TIFR Bombay, for his suggestions.

# References I


H. de Fraysseix, J. Pach, and R. Pollack.
How to draw a planar graph on a grid.
*Combinatorica*, 10, 1990.


S. Even, A. Lempel, and I. Cederbaum.
An algorithm for planarity testing of graphs.
In *Theory of Graphs: Internat. Symposium (Rome 1966)*, pages 215–232, Gordon and Breach, New York, 1967.


F. Harary.
*Graph Theory*.
Addison-Wesley, Reading, Mass., 1969.

# References II


F. Haray and W. T. Tutte.
A dual form of kuratowski's theorem.
*Bull. Amer. Math. Soc.*, 21(1):168, 1965.


J. Hopcroft and R. E. Tarjan.
Efficient planarity testing.
*Journal of ACM*, 21(4):549–568, 1974.


Kazimierz Kuratowski.
Sur le problème des courbes gauches en topologie.
*Fund. Math.*, 15:271–283, 1930.

# References III


Kurt Mehlhorn.
*Data Structures and Algorithms 2: Graph Algorithms and NP-Completeness, volume 2 of Monographs in Theoretical Computer Science*.
An EATCS Series. Springer-Verlag, 1984.


Kurt Mehlhorn and Petra Mutzel.
On the embedding phase of the hopcroft and tarjan planarity testing algorithm.
*Algorithmica*, 1996.


W. Schnyder.
Embedding planar graphs on the grid.
In *First ACM-SIAM Symp. on Discrete Algorithms*, pages 138–148, San Francisco, 1990.

# References IV

K. Wagner.
Uber eine erweiterung eines satzes von kuratowski.
*Deutsche Mathematik*, 2:280–285, 1937.

# At Last . . .

## Thank You

shreesh@rkmvu.ac.in
sarvottamananda@gmail.com