# Tangling and Untangling Trees on Point-sets

Giuseppe (Beppe) Liotta


University of Perugia, Italy

**Joint work with:** G. Di Battista, M. Patrignani, A. Symvonis, I. G. Tollis

Finanziato dall'Unione europea NextGenerationEU | Ministero dell'Università e della Ricerca | Italiadomani Piano Nazionale di Ripresa e Resilienza | unipg Università degli Studi di Perugia

RASTA

# The problem

Compute a simple drawing of a tree with a prescribed number of crossings on a given set S of points, while ensuring that its curve complexity is bounded by a constant

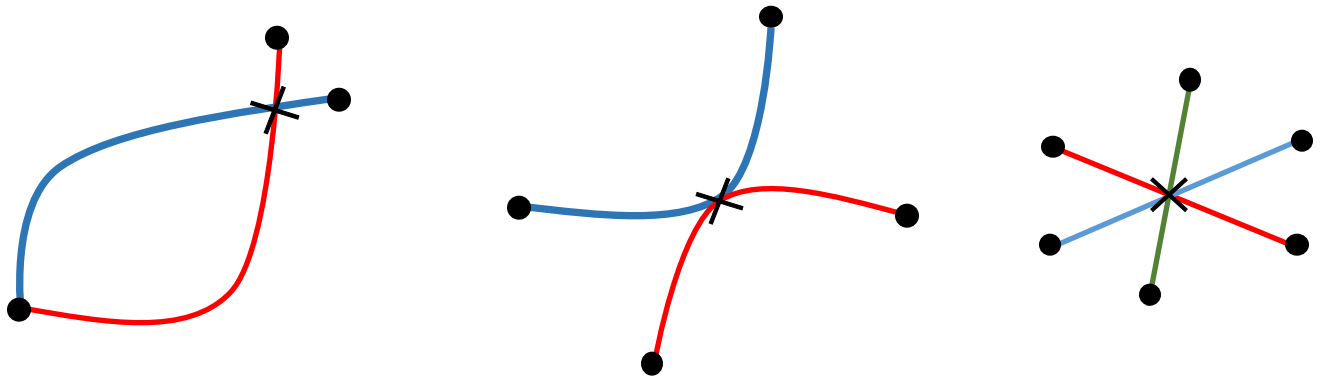
# The problem

Compute a simple drawing of a tree with a prescribed number of crossings on a given set S of points, while ensuring that its curve complexity is bounded by a constant

# The problem

Compute a simple drawing of a tree with a prescribed number of crossings on a given set S of points, while ensuring that its curve complexity is bounded by a constant

any two edges can share at most one interior point (i.e. a crossing) and the following configurations are forbidden



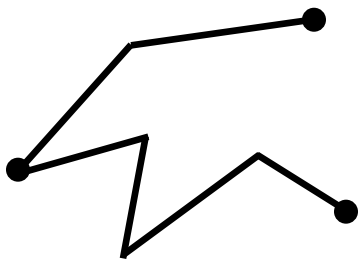
# The problem

Compute a simple drawing of a tree with a prescribed number of crossings on a given set S of points, while ensuring that its curve complexity is bounded by a constant

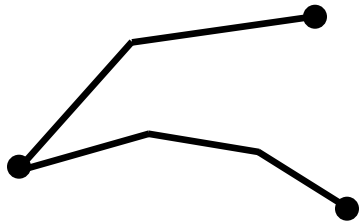
# The problem

Compute a simple drawing of a tree with a prescribed number of crossings on a given set S of points, while ensuring that its curve complexity is bounded by a constant

when edges are represented as polygonal chains, the curve complexity is the maximum number of bends per edge



curve complexity : 3



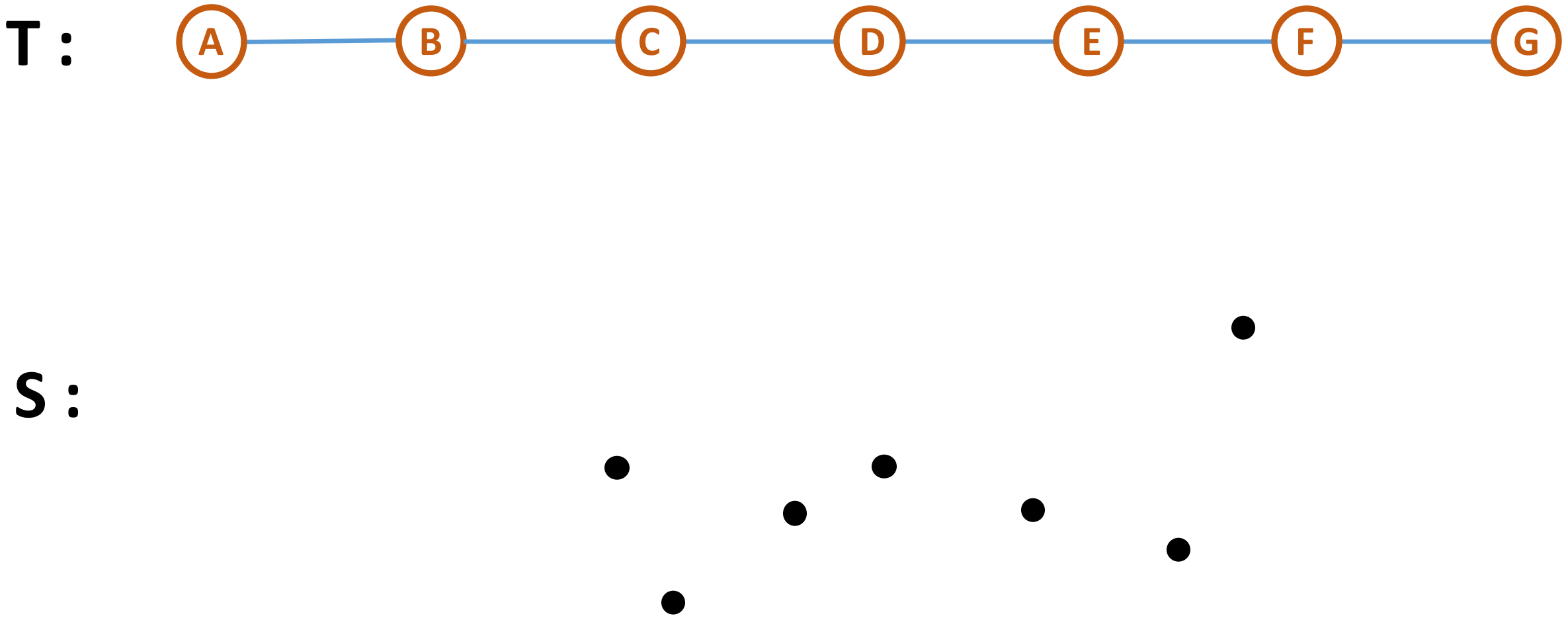
curve complexity : 2

# Example

# Example

**T :**

# Example

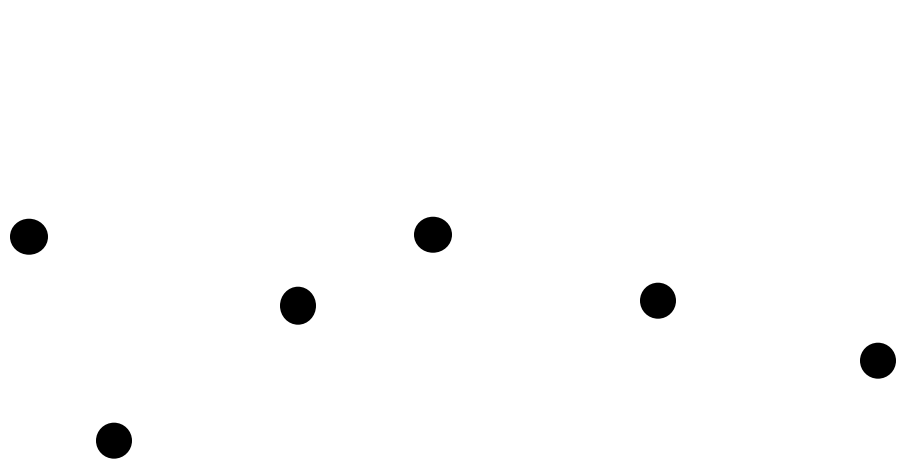


# Example

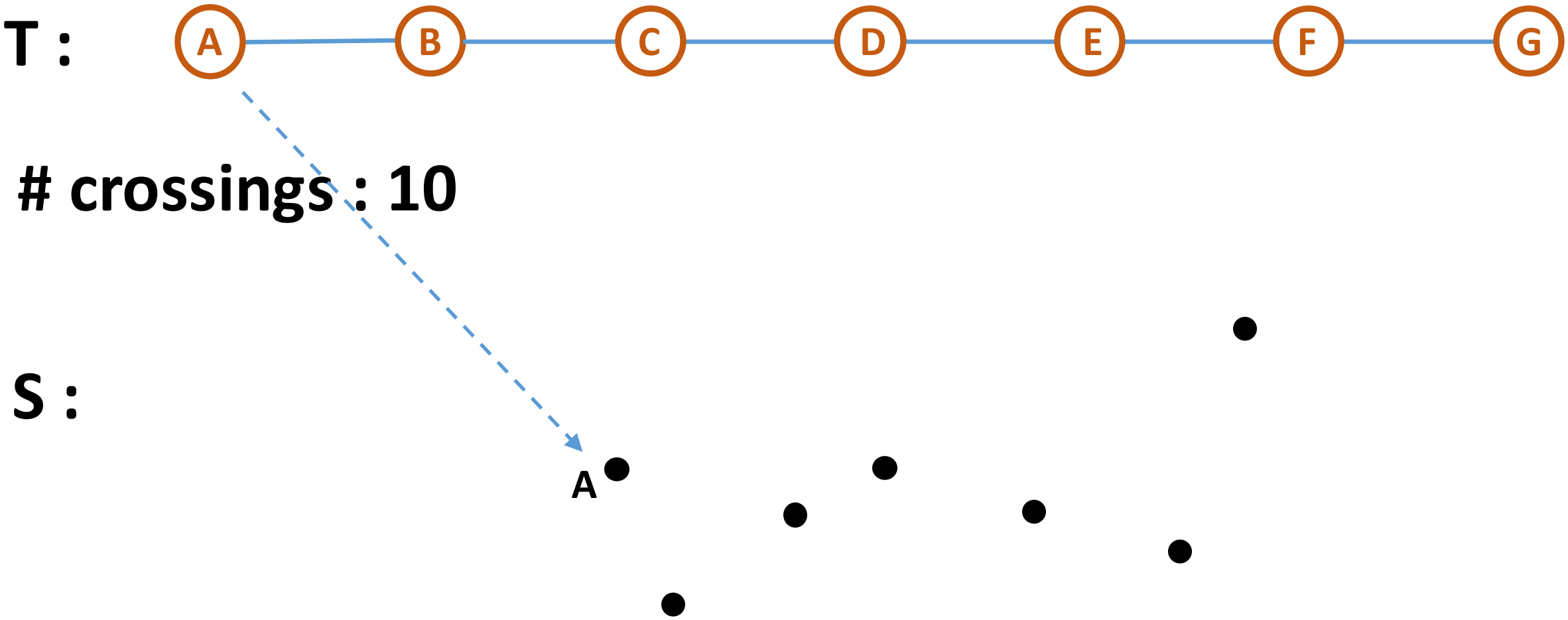
**T :** 


**# crossings : 10**

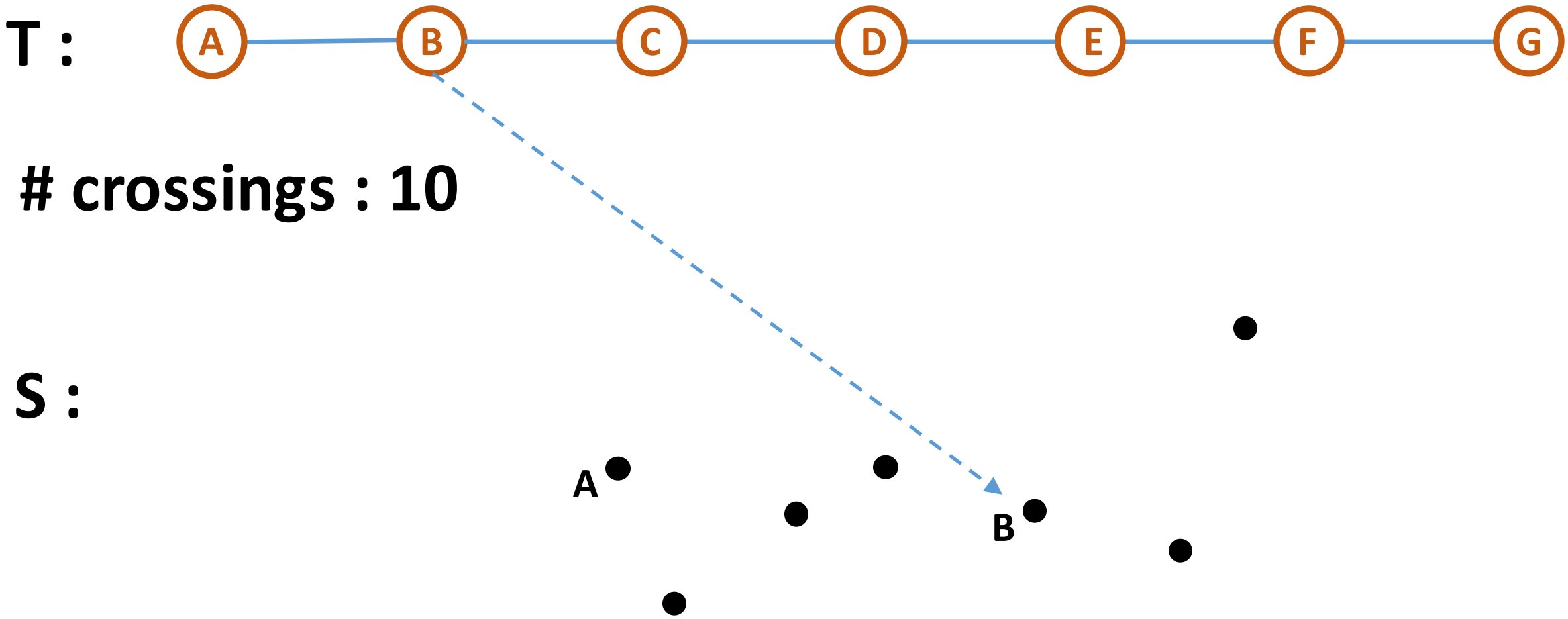
**S :**



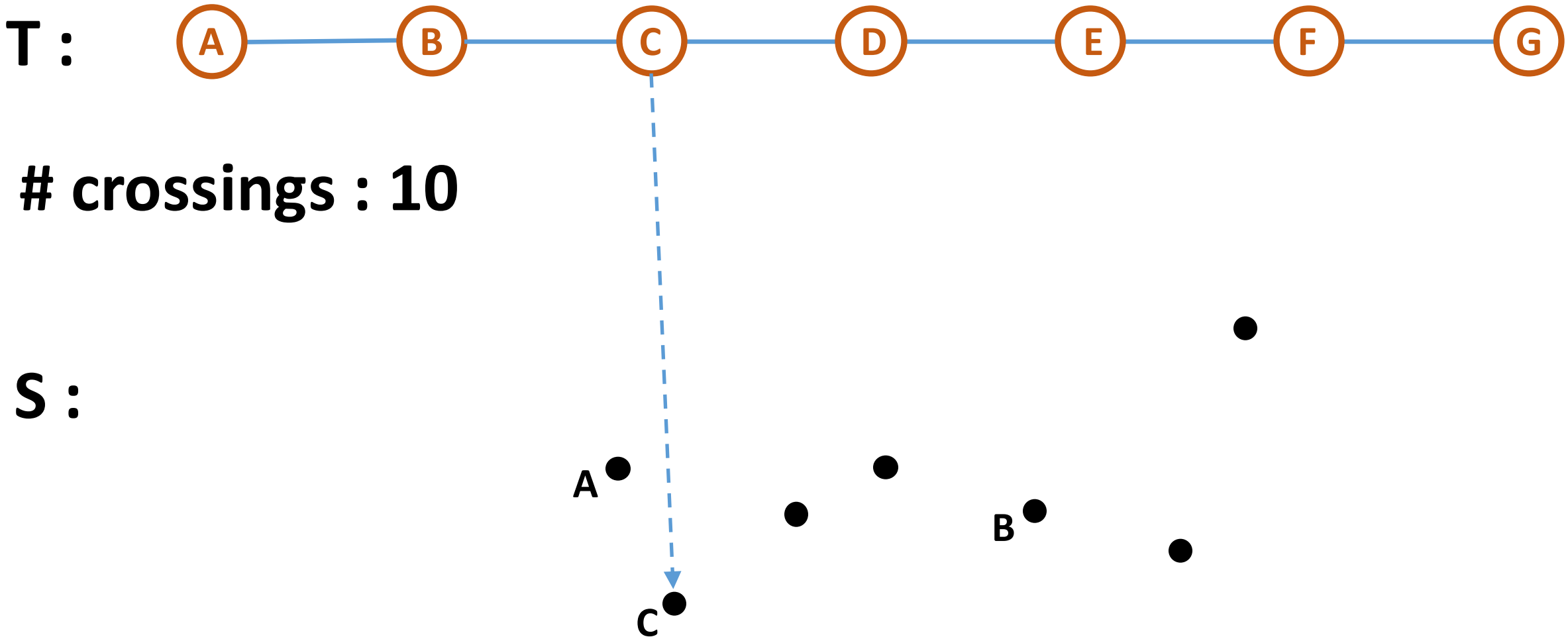
# Example



# Example



# Example

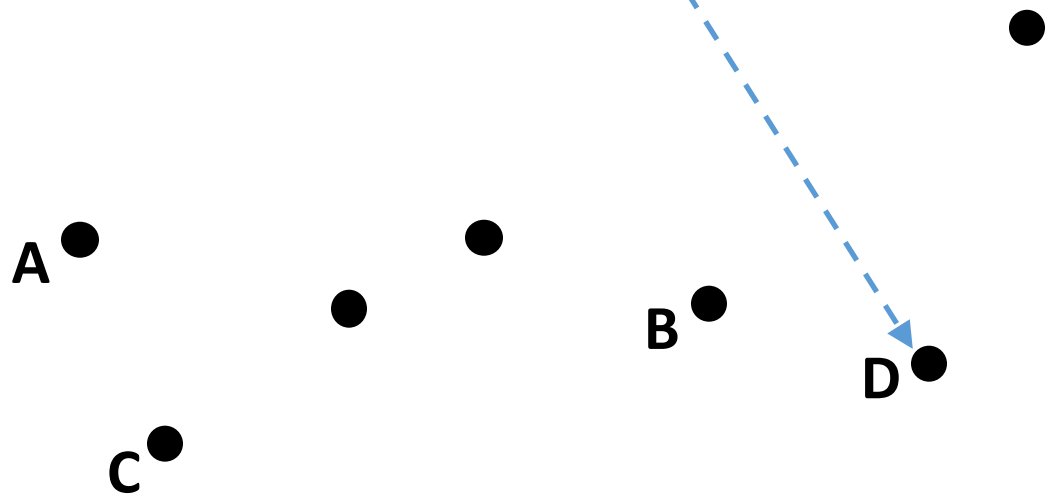


# Example

**T :**


**# crossings : 10**

**S :**

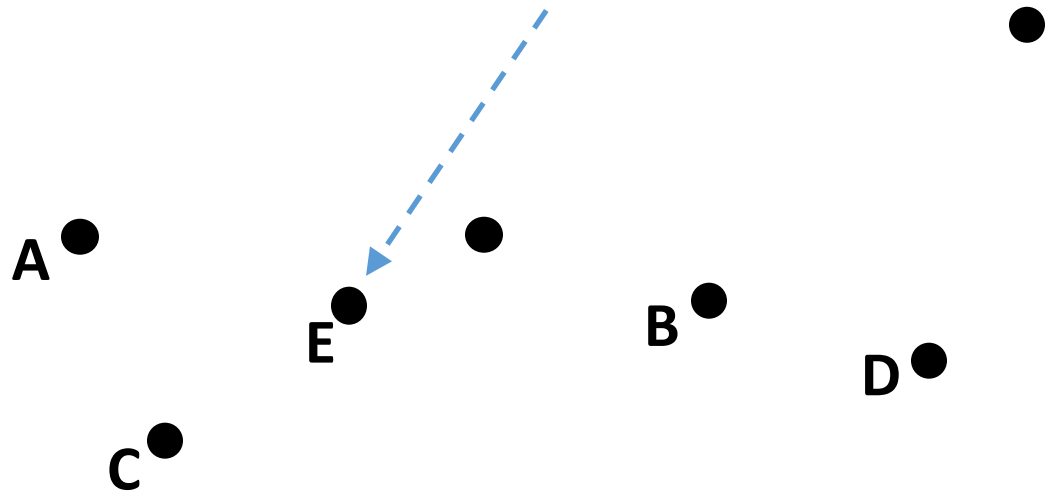


# Example

**T :**


**# crossings : 10**

**S :**

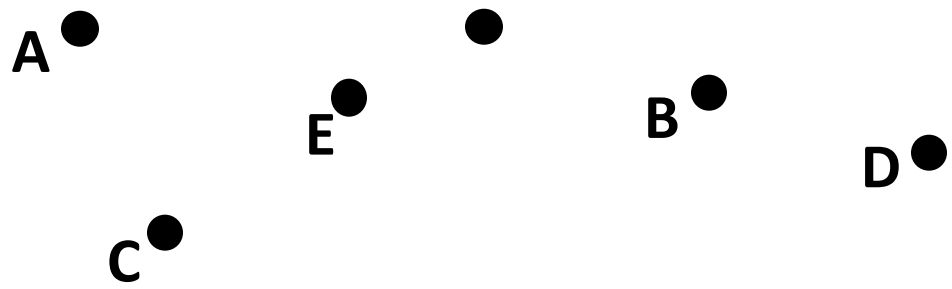


# Example

**T :**


**# crossings : 10**

**S :**

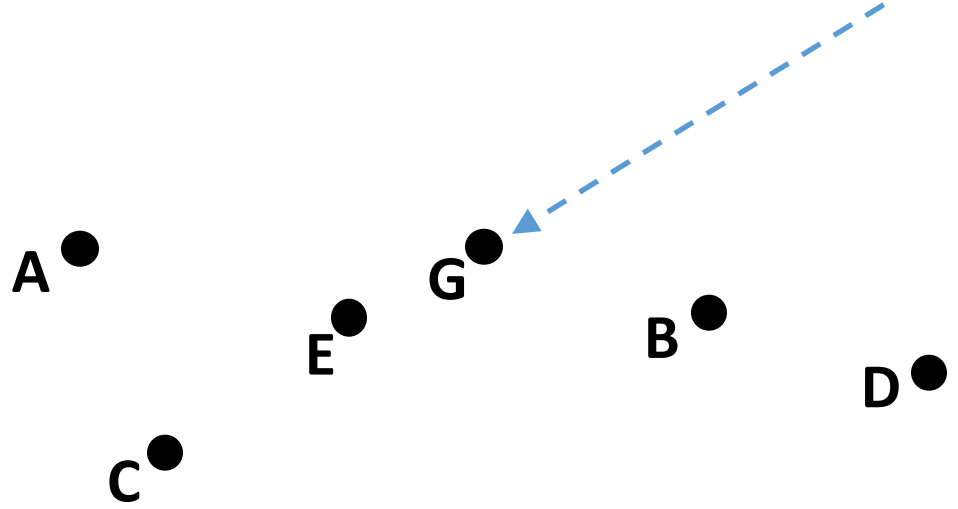


# Example

**T :**


**# crossings : 10**

**S :**

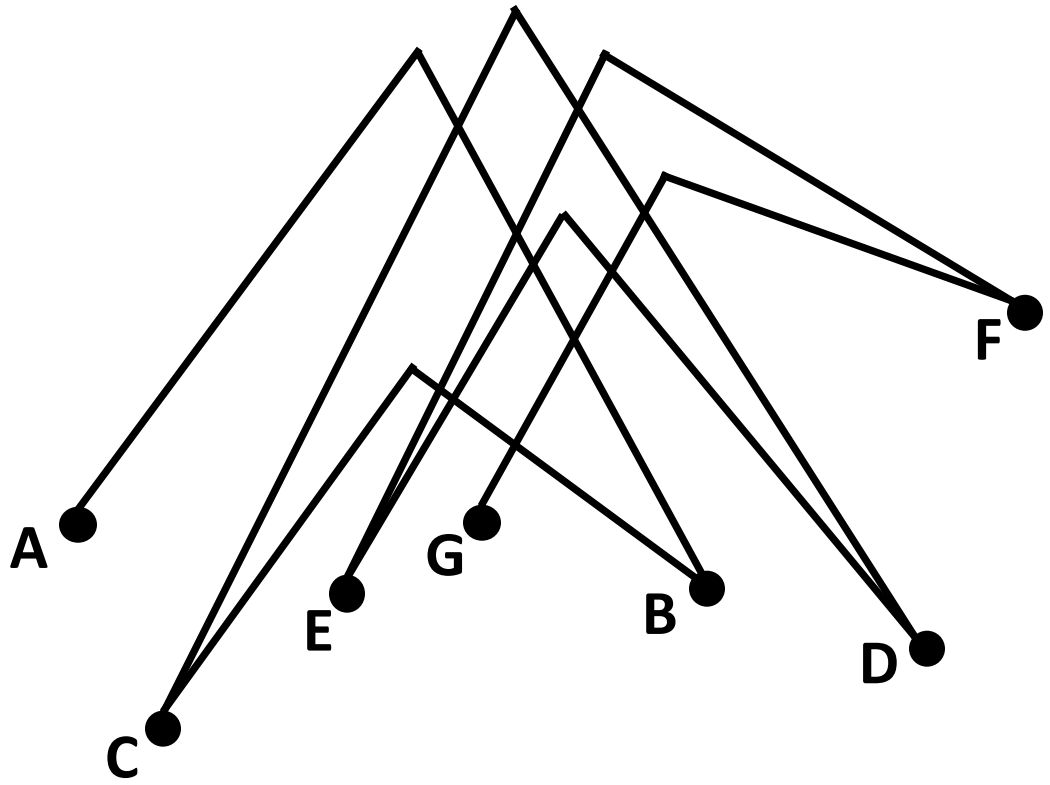


# Example

**T :**


**# crossings : 10**

**S :**

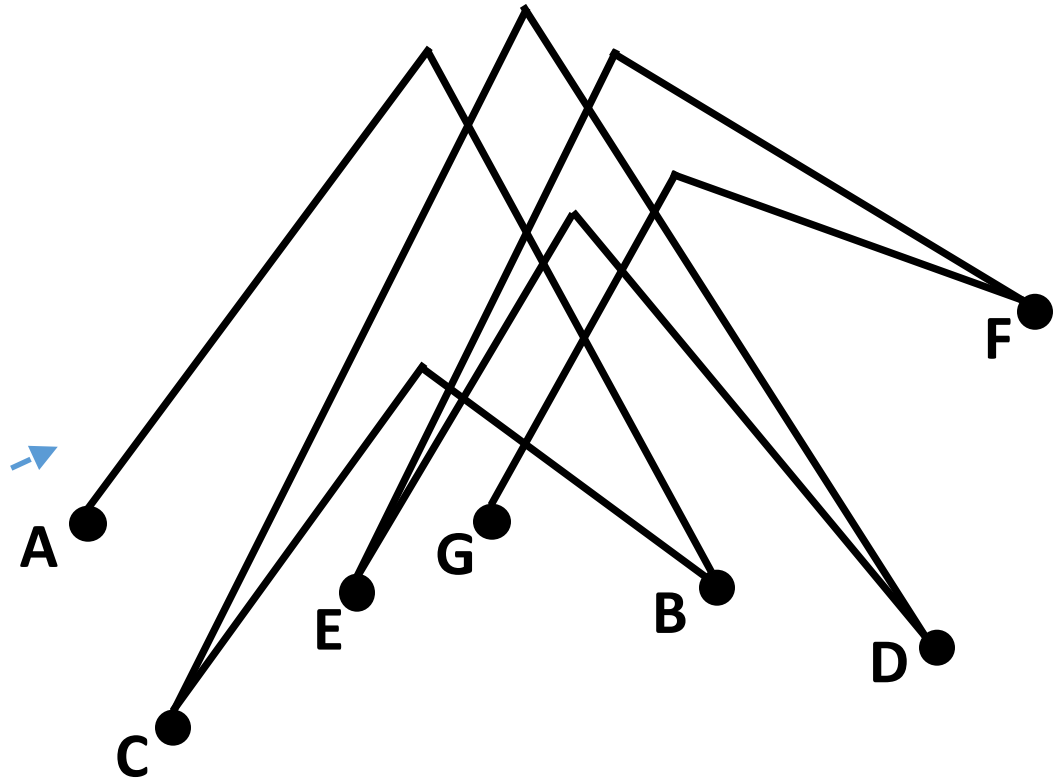


# Example

**T :**


**# crossings : 10**

**S :**



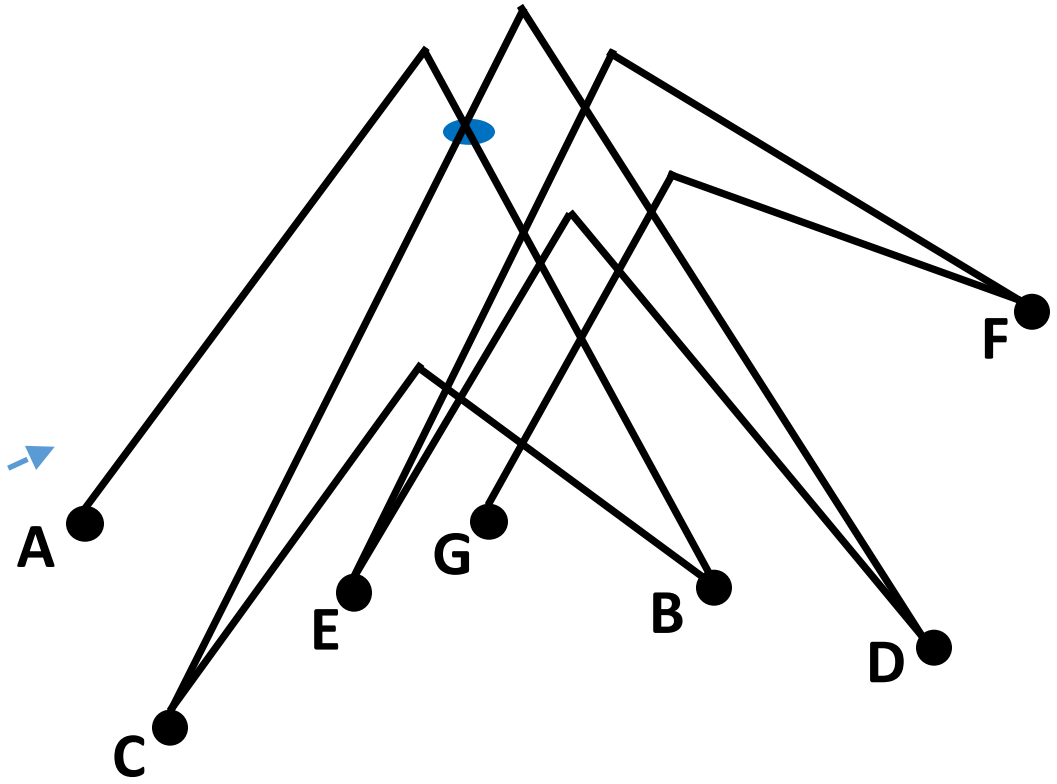

Point set embedding of T on S

# Example

**T :**


**# crossings : 10**

**S :**




1

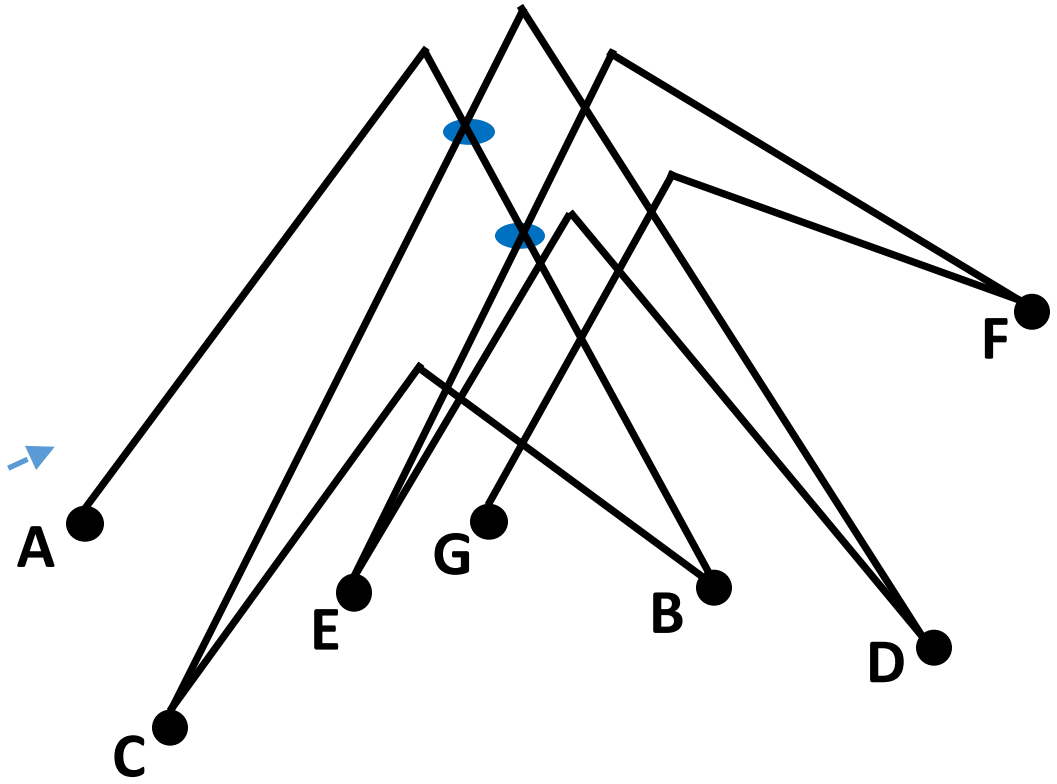
Point set embedding of T on S

# Example

**T :**


**# crossings : 10**

**S :**




2

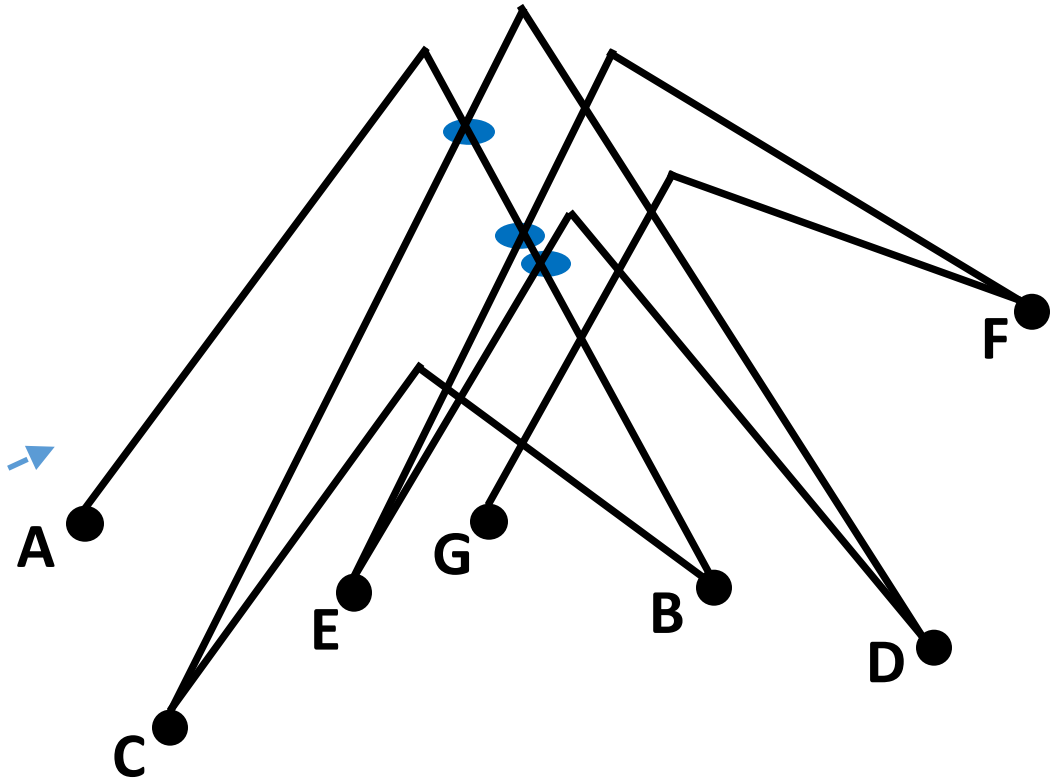
Point set embedding of T on S

# Example

**T :**


**# crossings : 10**

**S :**




3

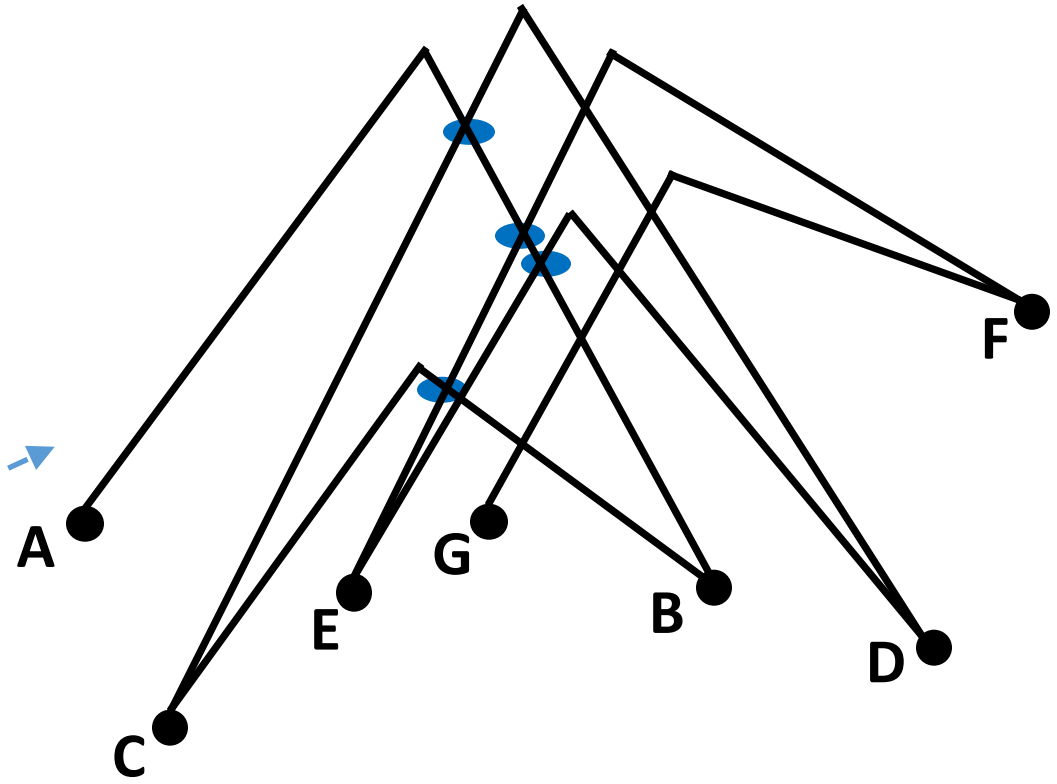
Point set embedding of T on S

# Example

**T :**


**# crossings : 10**

**S :**




4

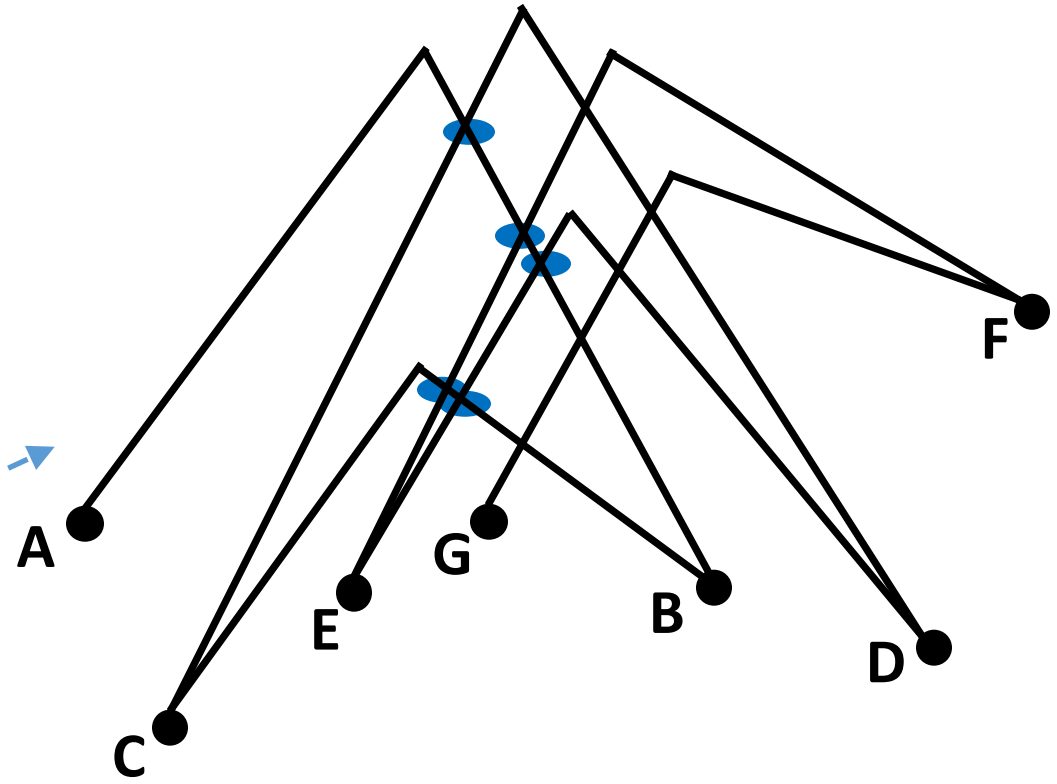
Point set embedding of T on S

# Example

**T :**


**# crossings : 10**

**S :**




5

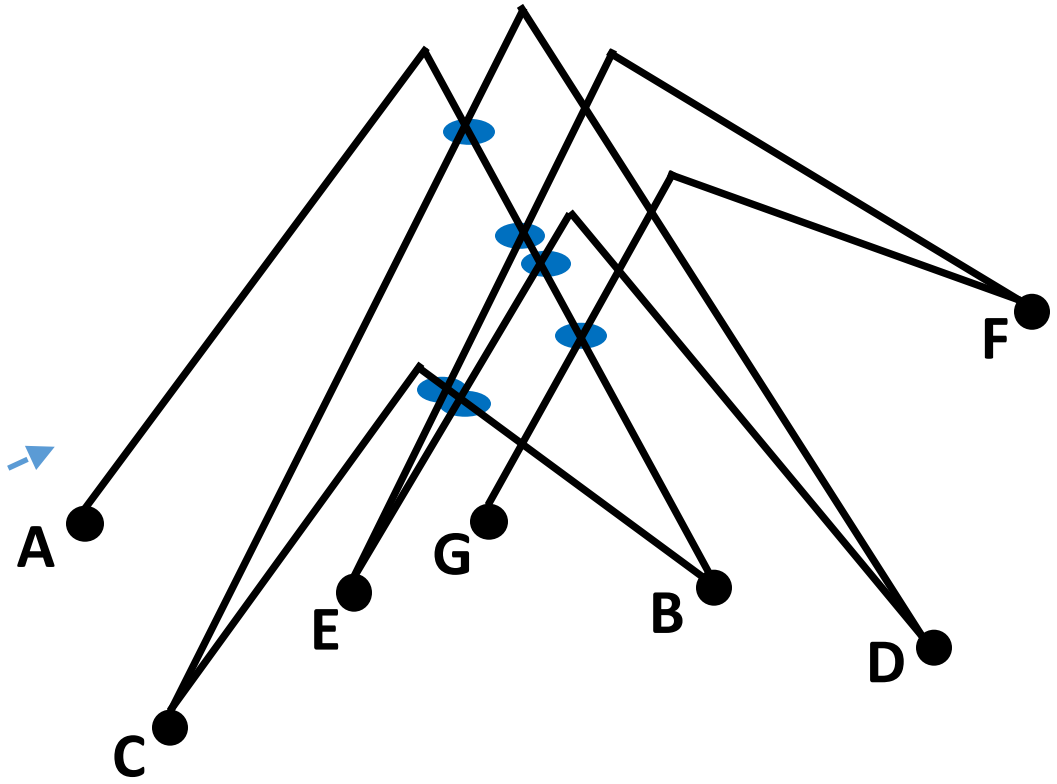
Point set embedding of T on S

# Example

**T :**


**# crossings : 10**

**S :**




6

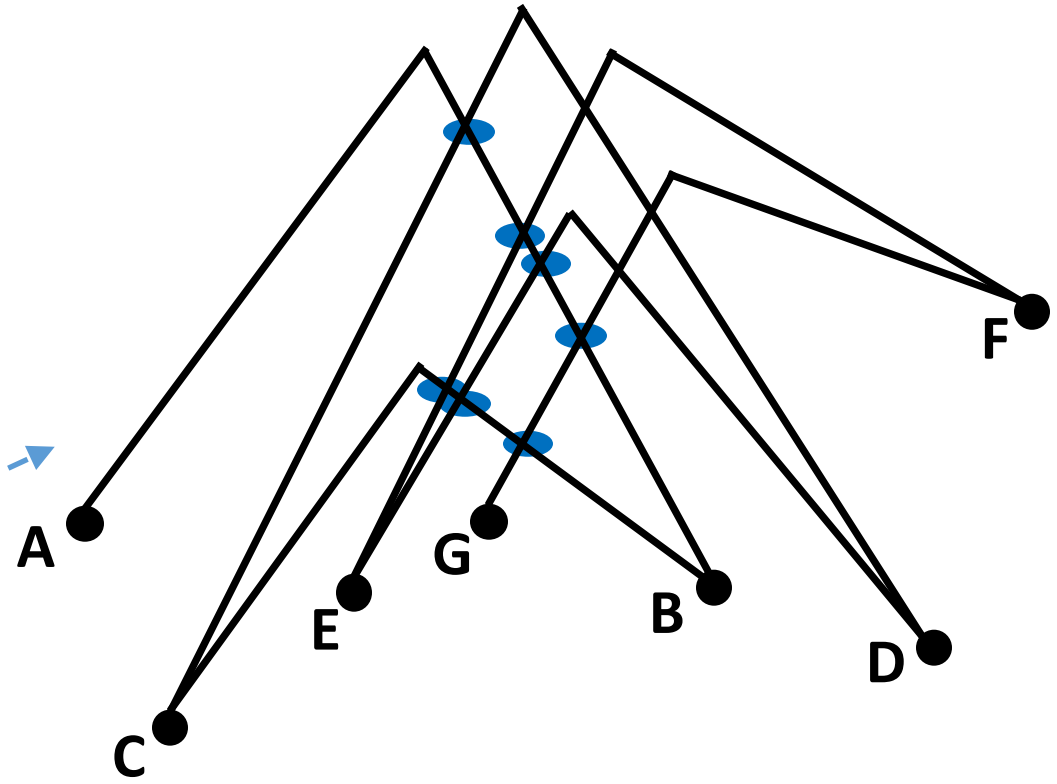
Point set embedding of T on S

# Example

**T :**


**# crossings : 10**

**S :**




7

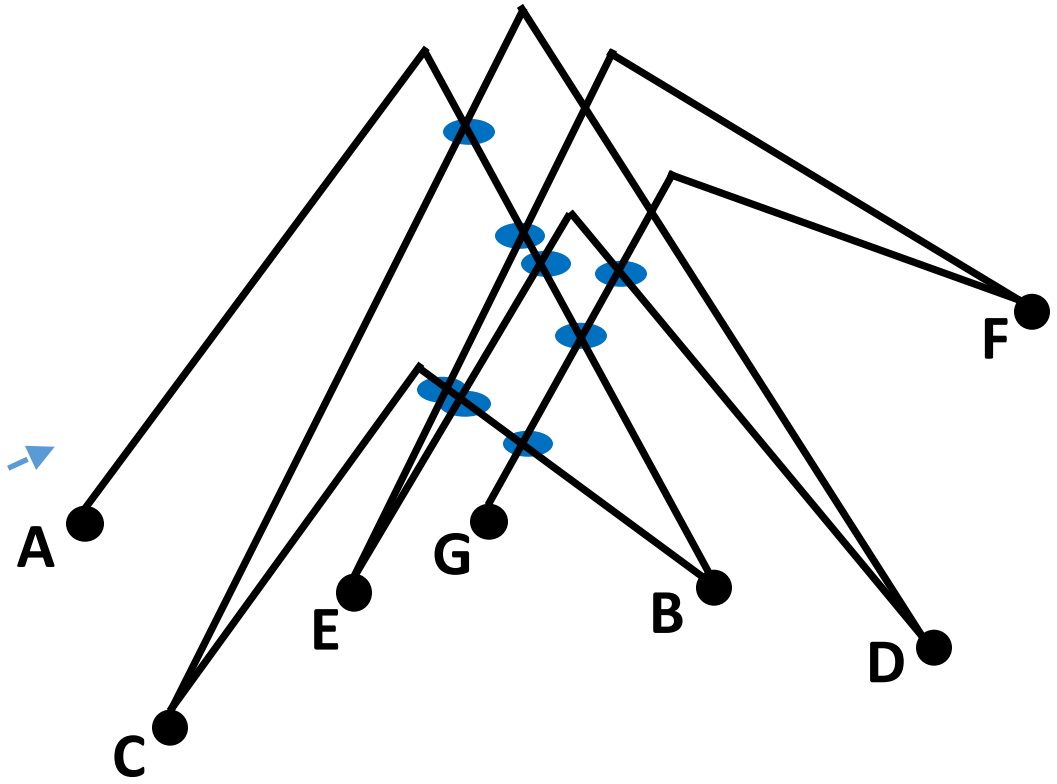
Point set embedding of T on S

# Example

**T :**


**# crossings : 10**

**S :**



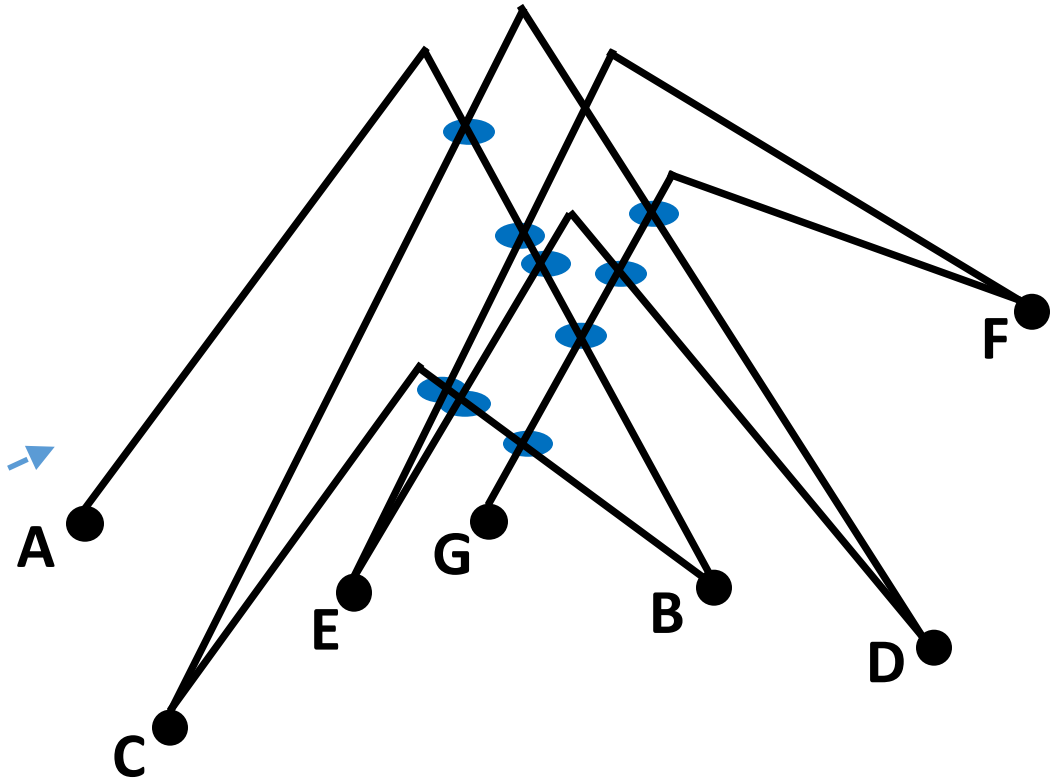

Point set embedding of T on S

# Example

**T :**


**# crossings : 10**

**S :**



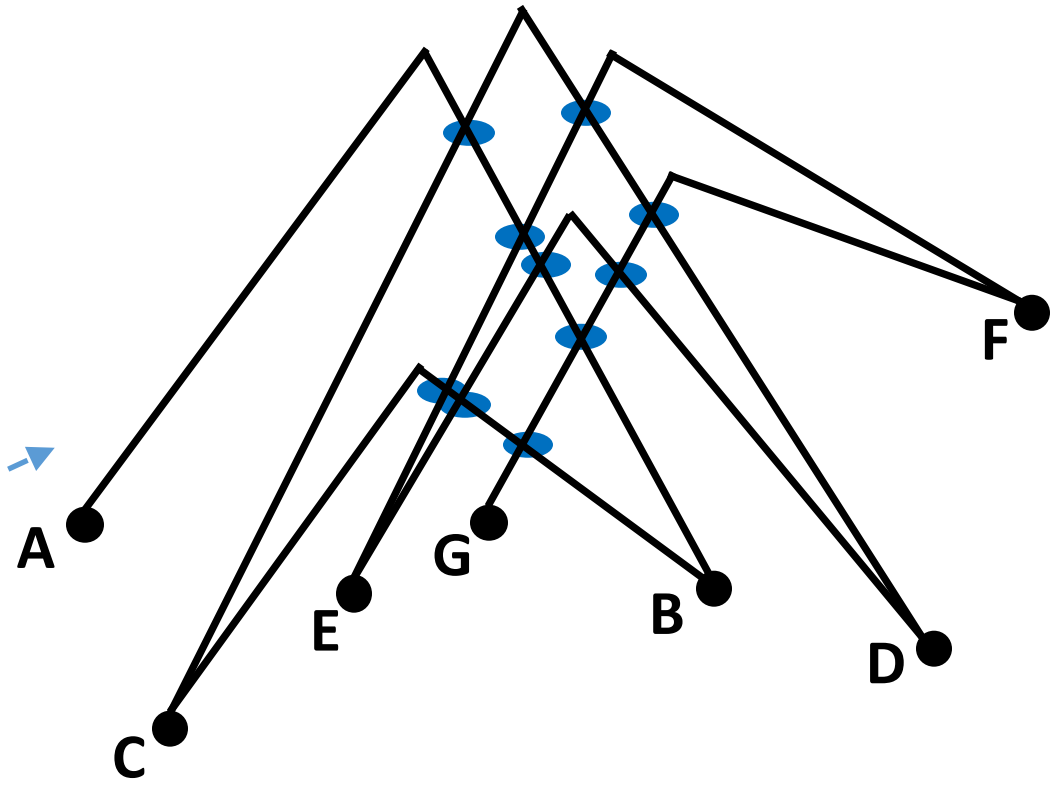

Point set embedding of T on S

# Example

**T :**


**# crossings : 10**

**S :**




10

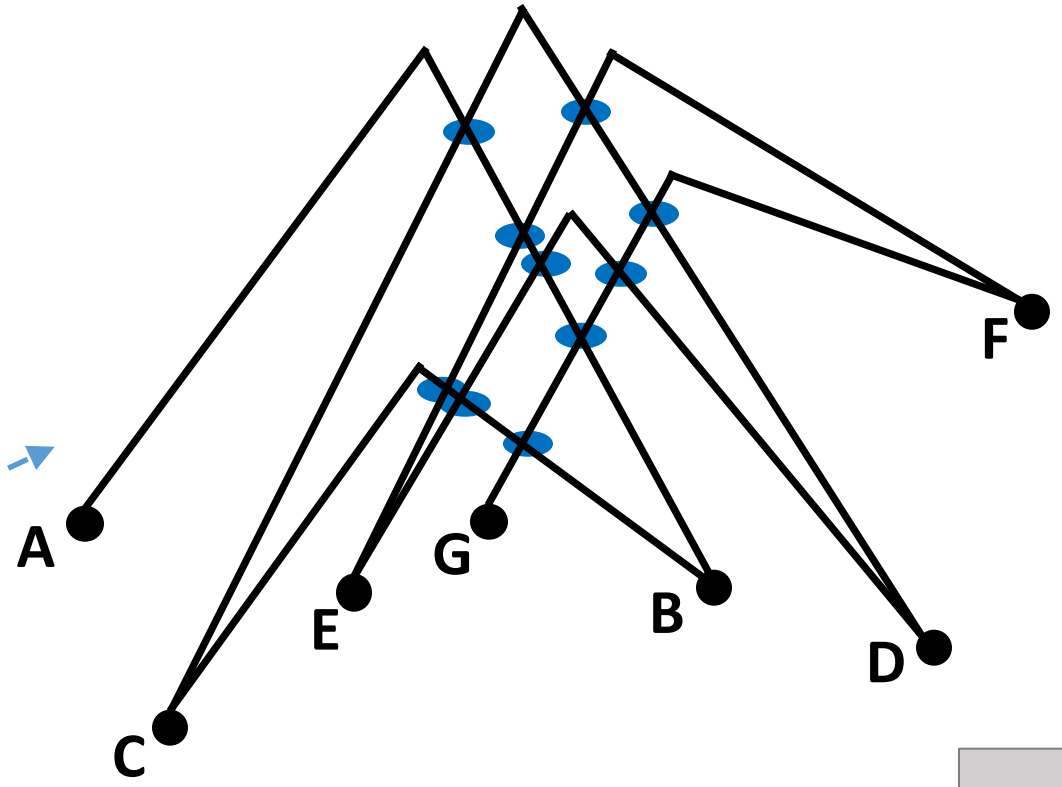
Point set embedding of T on S

# Example

**T :**


**# crossings : 10**

**S :**




10

Point set embedding of T on S

curve complexity : 1

# Thrackles

- The *thrackle bound* $\vartheta(G)$ of a graph G = (V,E) with m edges is the number of crossings that a simple drawing of G would have if every edge can cross every other non-adjacent edge. It is known that

$$\vartheta(G) = \frac{1}{2}(m(m+1) - \sum_{v \in v} \deg(v)^2)$$

# Thrackles

- The *thrackle bound* $\vartheta(G)$ of a graph G = (V,E) with m edges is the number of crossings that a simple drawing of G would have if every edge can cross every other non-adjacent edge. It is known that

$$\vartheta(G) = \frac{1}{2}(m(m+1) - \sum_{v \in v} \deg(v)^2)$$

- A graph is *thrackable* if it admits a simple drawing with $\vartheta(G)$ edge crossings

# Thrackles

- The *thrackle bound* $\vartheta(G)$ of a graph G = (V,E) with m edges is the number of crossings that a simple drawing of G would have if every edge can cross every other non-adjacent edge. It is known that

$$\vartheta(G) = \frac{1}{2}(m(m+1) - \sum_{v \in v} \deg(v)^2)$$

- A graph is *thrackable* if it admits a simple drawing with $\vartheta(G)$ edge crossings

- Piazza, Ringeisen, and Stueckle prove that for every tree T and any integer $0 \leq \chi \leq \vartheta(T)$, T admits a simple drawing with $\chi$ crossings

# Thrackles

- The *thrackle bound* $\vartheta(G)$ of a graph G = (V,E) with m edges is the number of crossings that a simple drawing of G would have if every edge can cross every other non-adjacent edge. It is known that

$$\vartheta(G) = \frac{1}{2}(m(m+1) - \sum_{v \in v} \deg(v)^2)$$

- A graph is *thrackable* if it admits a simple drawing with $\vartheta(G)$ edge crossings

- Piazza, Ringeisen, and Stueckle prove that for every tree T and any integer $0 \leq \chi \leq \vartheta(T)$, T admits a simple drawing with $\chi$ crossings; curve complexity may be O(n)

# Thrackles

- The *thrackle bound* $\vartheta(G)$ of a graph G = (V,E) with m edges is the number of crossings that a simple drawing of G would have if every edge can cross every other non-adjacent edge. It is known that

$$\vartheta(G) = \frac{1}{2}(m(m+1) - \sum_{v \in v} \deg(v)^2)$$

- A graph is *thrackable* if it admits a simple drawing with $\vartheta(G)$ edge crossings

- Piazza, Ringeisen, and Stueckle prove that for every tree T and any integer $0 \leq \chi \leq \vartheta(T)$, T admits a simple drawing with $\chi$ crossings; curve complexity may be O(n); no fixed location for the vertices

# Our contribution

We present and $O(n^2)$-time algorithm to compute a point set embedding (pse) of a tree T on any set S of points with curve complexity O(1) and any number of crossings in the range $0 \leq \chi \leq \vartheta(T)$

# Our contribution

We present and $O(n^2)$-time algorithm to compute a point set embedding (pse) of a tree T on any set S of points with curve complexity O(1) and any number of crossings in the range $0 \leq \chi \leq \vartheta(T)$

More precisely:

- Generic pse: curve complexity 5, it reduces to 1 if T is a path;
- RAC pse: curve complexity 9, it reduces to 3 if T is a path;
- Also, the time complexity reduces to O(n log n) if T is a path.

# Our contribution

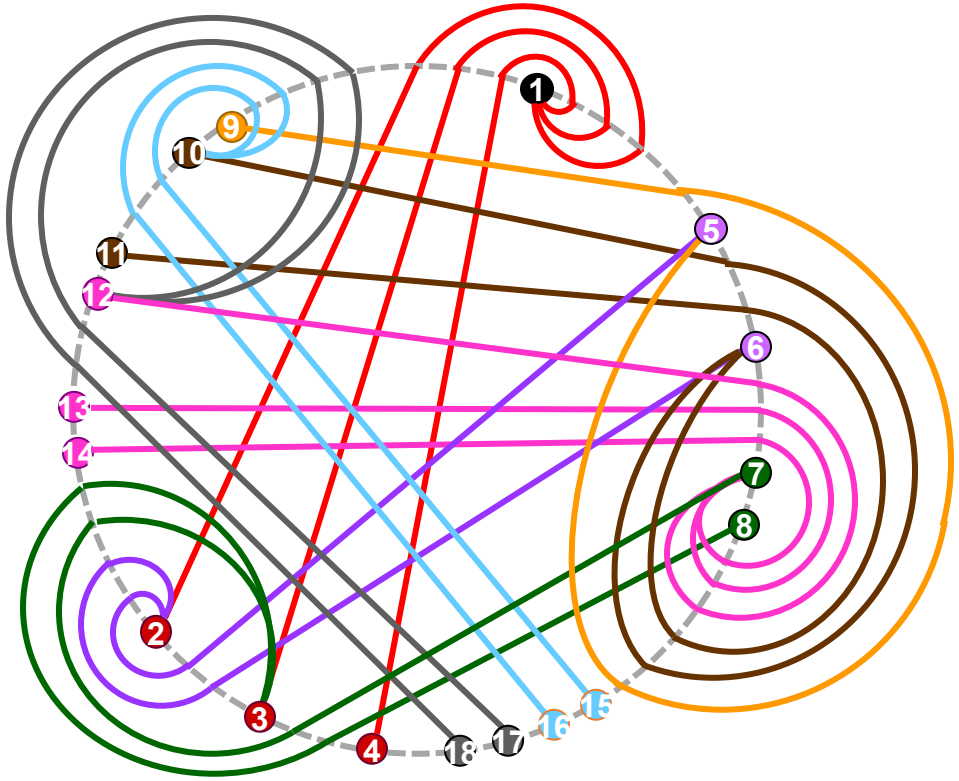
We present and $O(n^2)$-time algorithm to compute a point set embedding (pse) of a tree T on any set S of points with curve complexity O(1) and any number of crossings in the range $0 \leq \chi \leq \vartheta(T)$

More precisely:

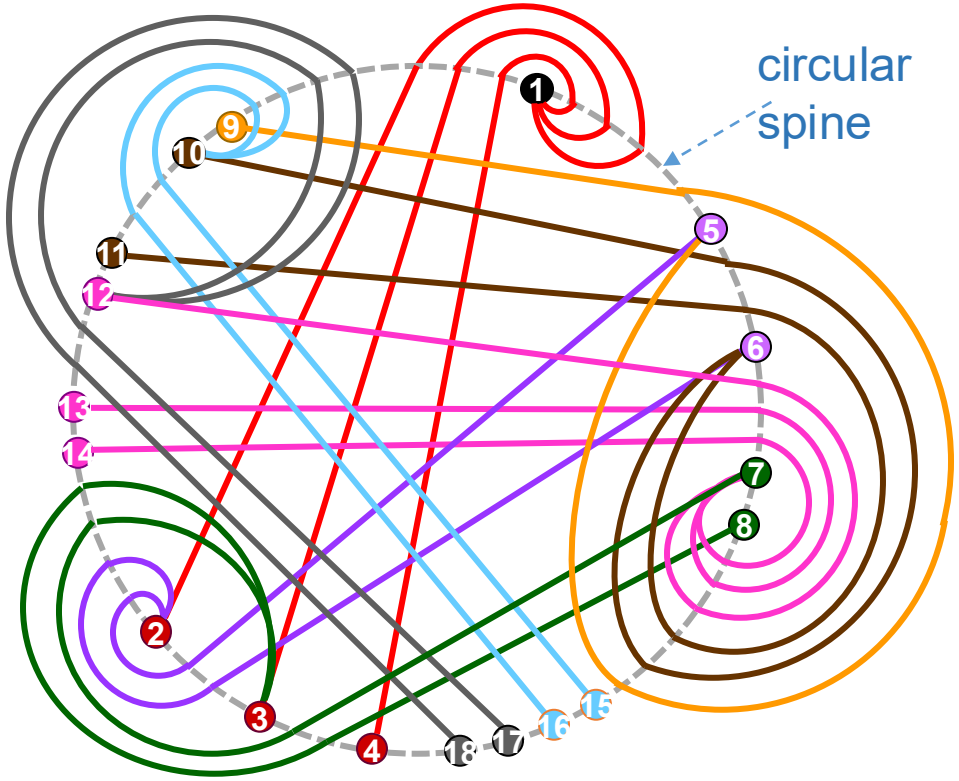
- Generic pse: curve complexity 5, it reduces to 1 if T is a path;
- RAC pse: curve complexity 9, it reduces to 3 if T is a path;
- Also, the time complexity reduces to O(n log n) if T is a path.

*Key ingredient:* efficient computation of a ***topological circular layout*** of a tree with $\chi$ crossings and O(1) circular spine traversals

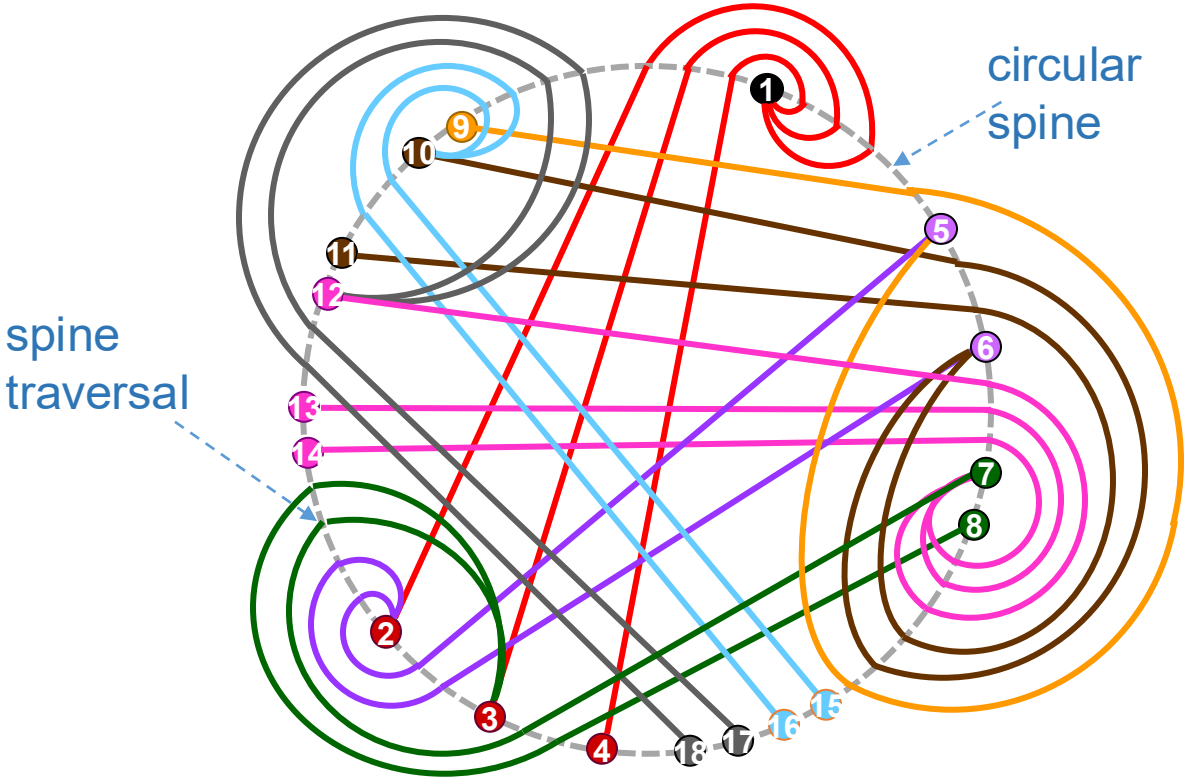
# Topological circular layout



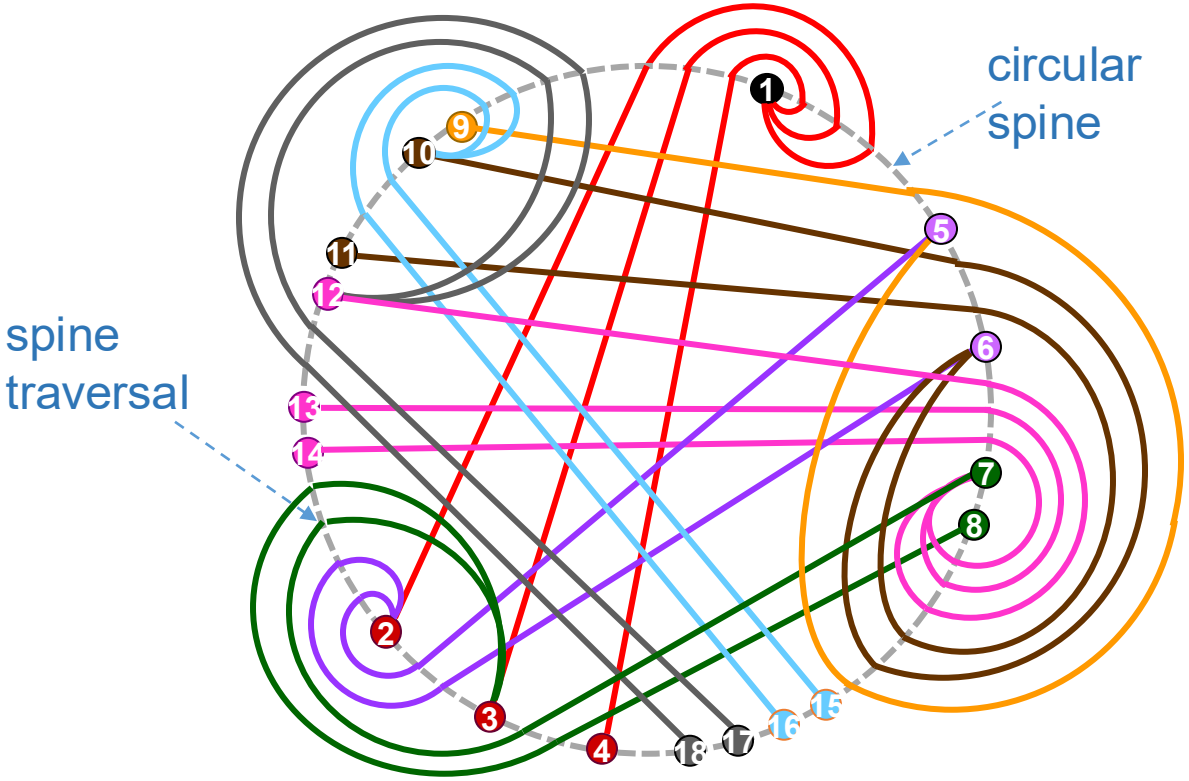
# Topological circular layout



# Topological circular layout



# Topological circular layout




If a connected graph with m edges admits a topological circular layout with O(1) circular spine traversals per edge, then it admits a topology preserving point set embedding with O(1) curve complexity which can be computed in O(m log m + T(m)) time, where T(m) is the time to compute the circular layout

# Circular layouts of trees with a prescribed number of crossings and at most 2 circular spine traversals per edge

# The tangle-untangle algorithm

# The tangle-untangle algorithm

*Tangling phase*: compute a topological circular layout that reaches the thrackle bound $\vartheta(T)$ and has most two circular spine traversals per edge

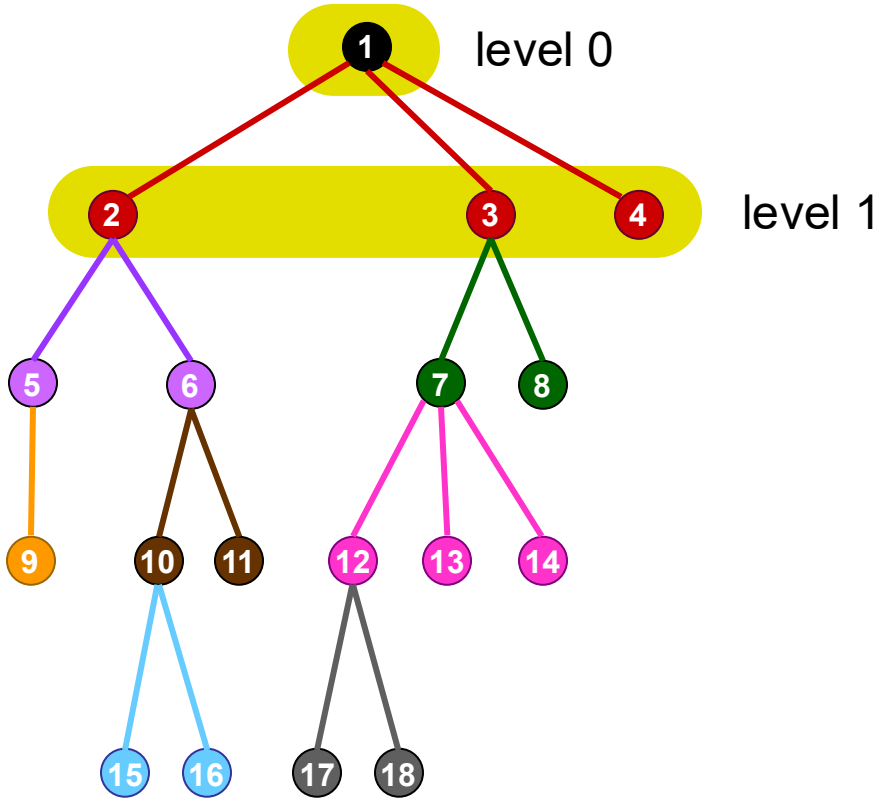
# The tangle-untangle algorithm

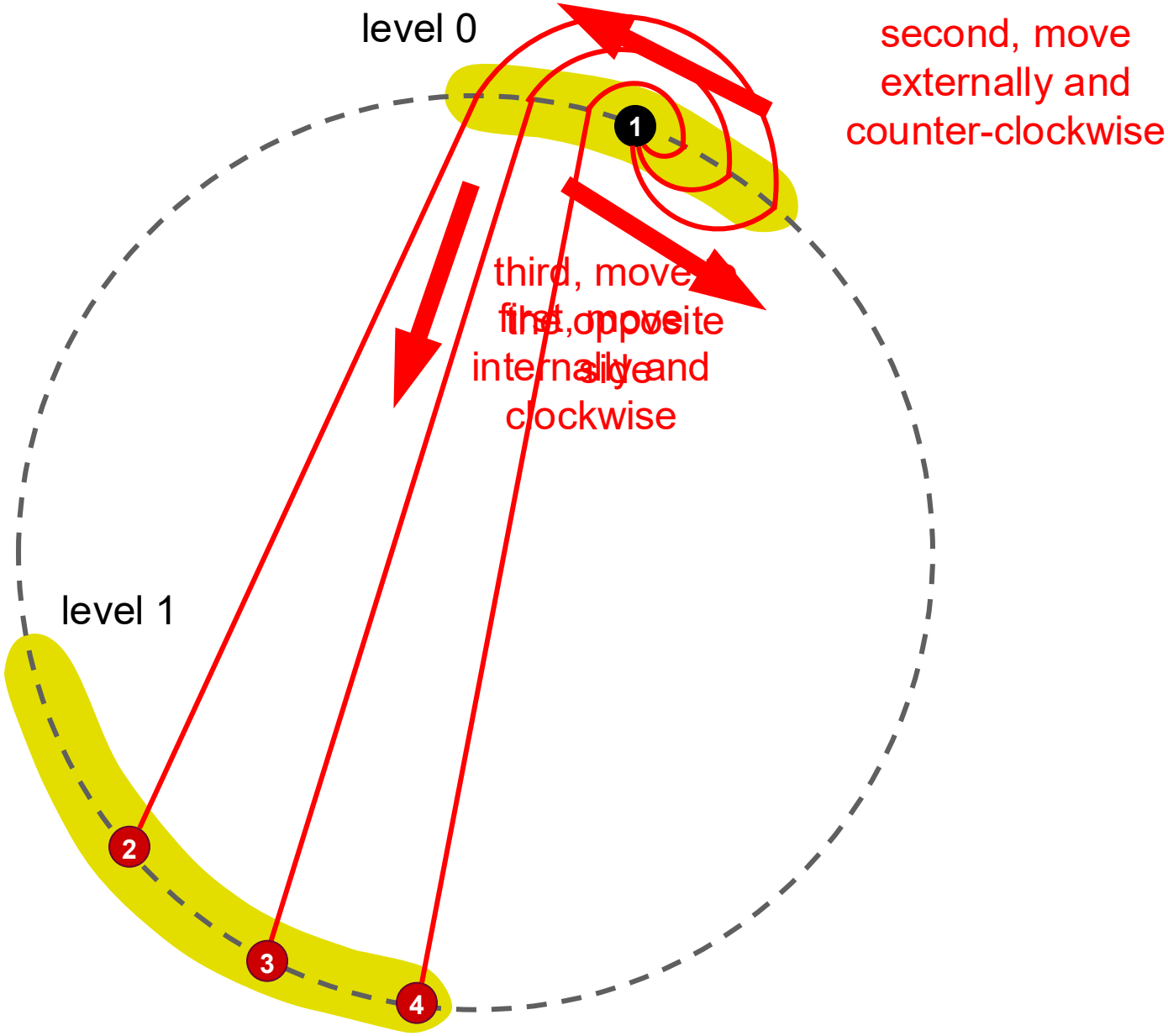
*Tangling phase*: compute a topological circular layout that reaches the thrackle bound $\vartheta(T)$ and has most two circular spine traversals per edge

*Untangling phase*: reduce the number of crossings in the topological circular layout one by one without increasing the number of circular spine traversals per edge

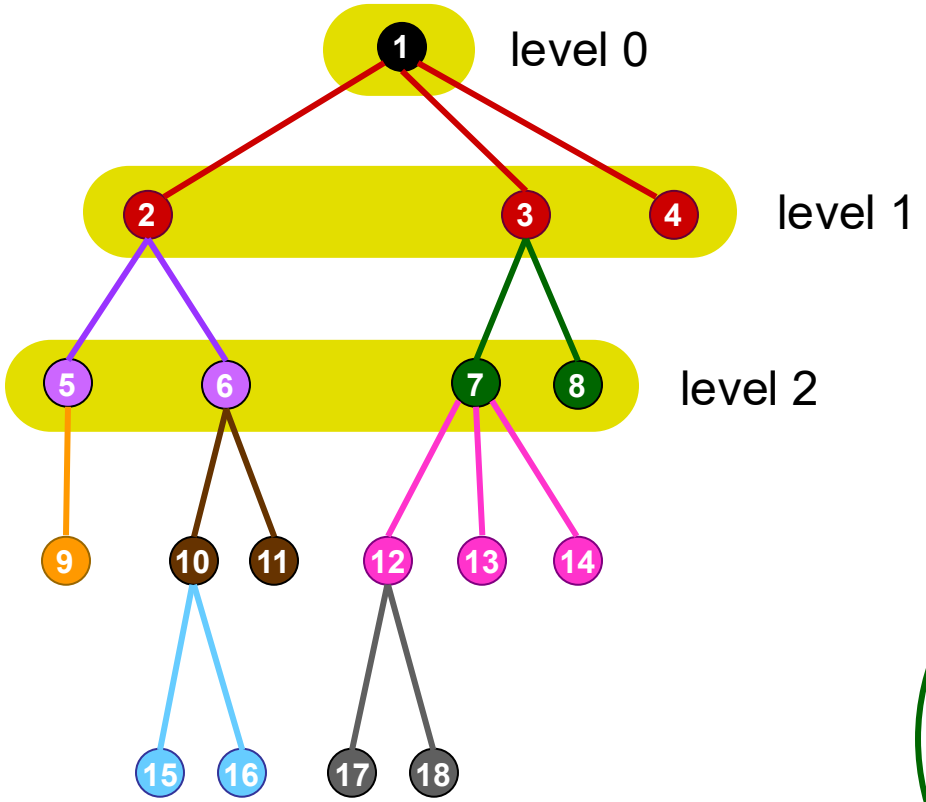
# The tangling phase

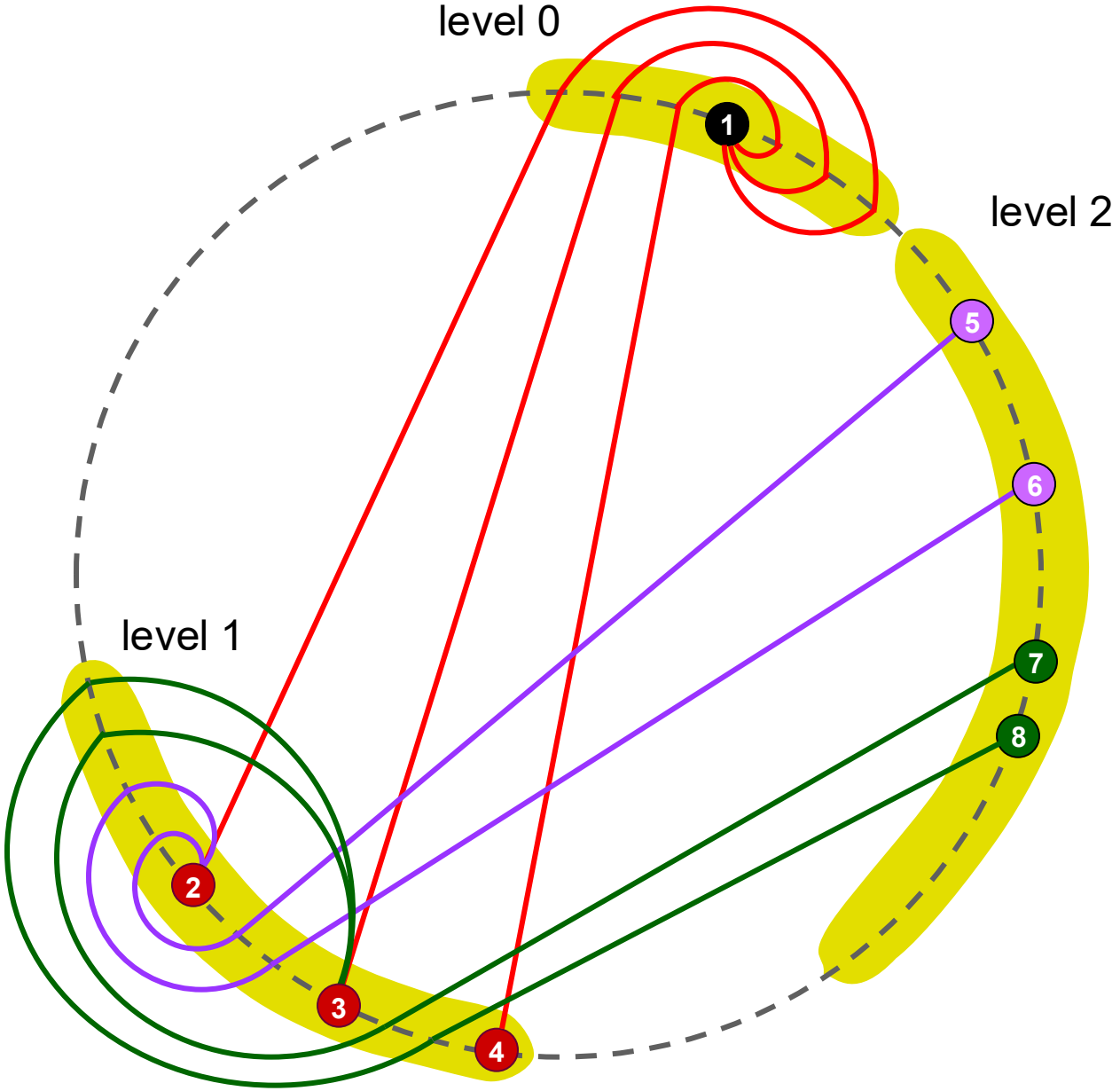
# How to tangle



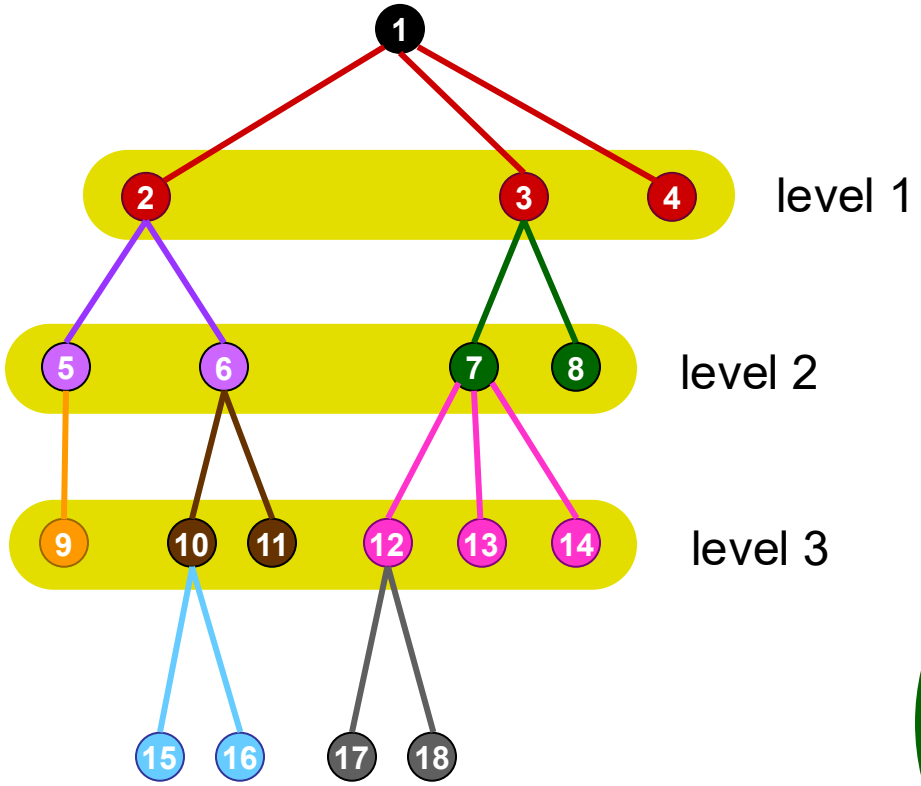



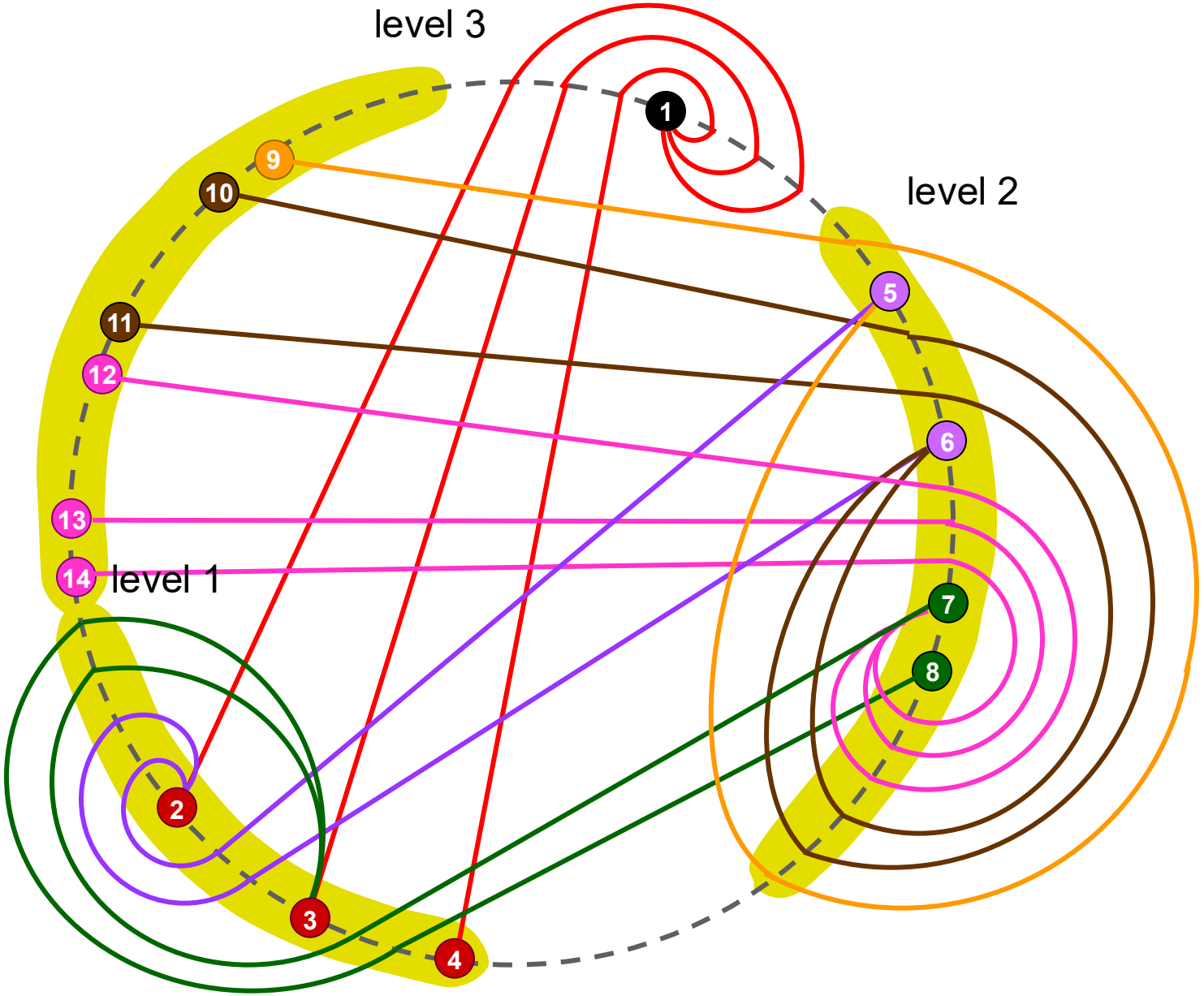
# How to tangle



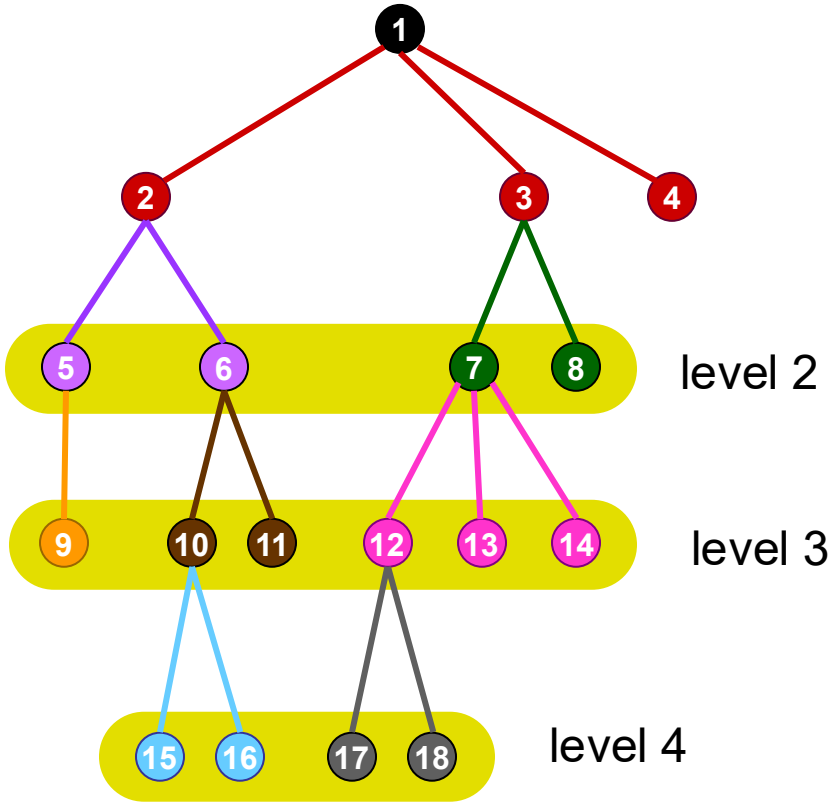



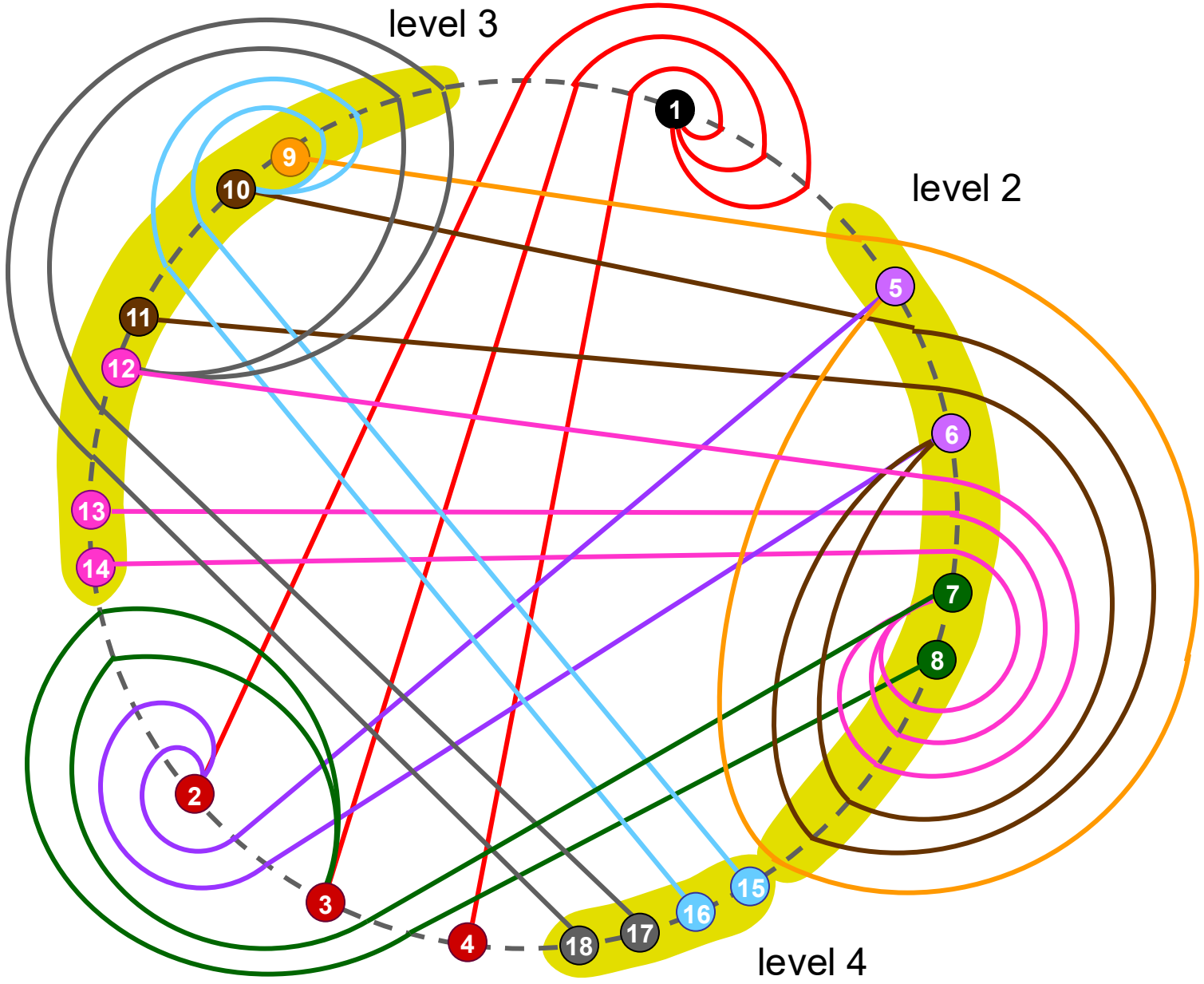
# How to tangle



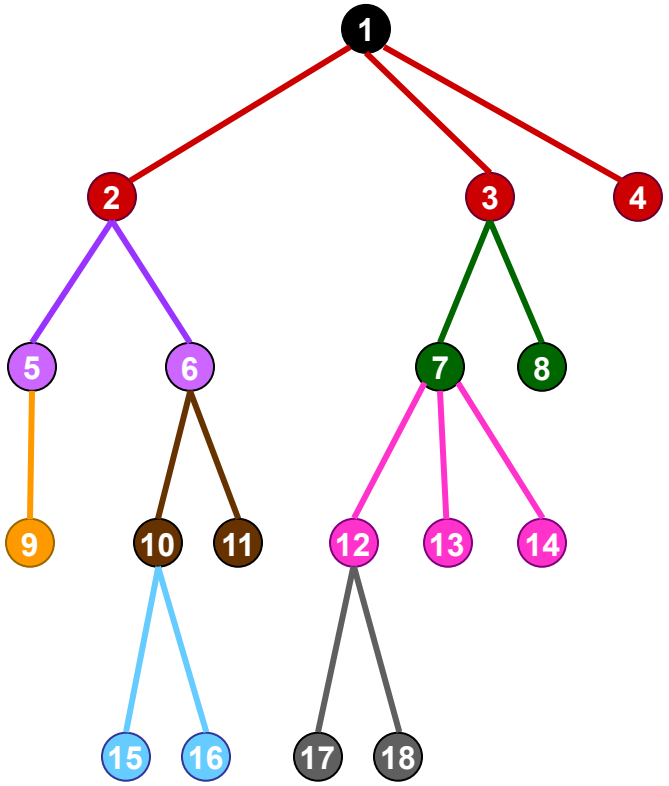



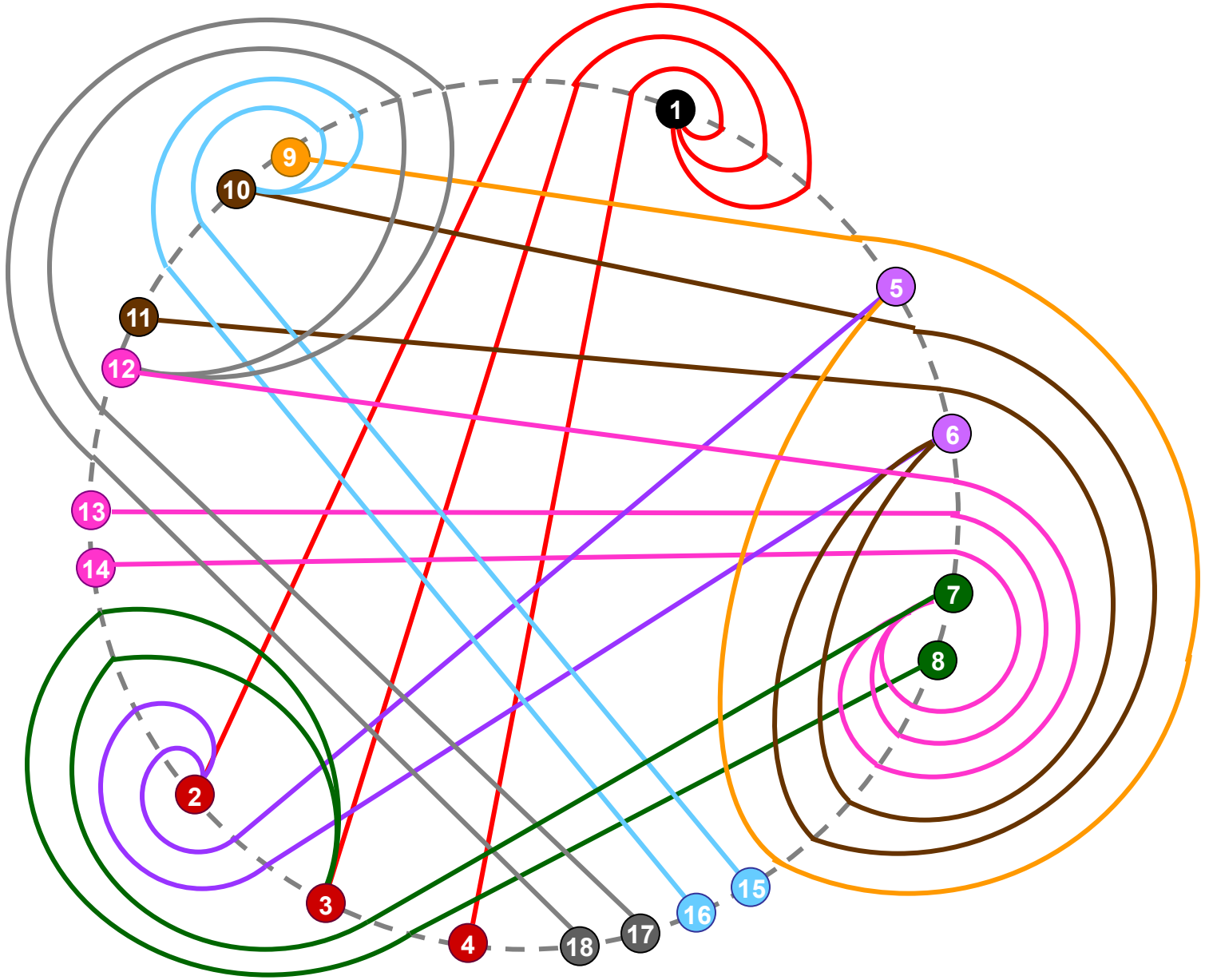
# How to tangle



# How to tangle



# The untangling phase

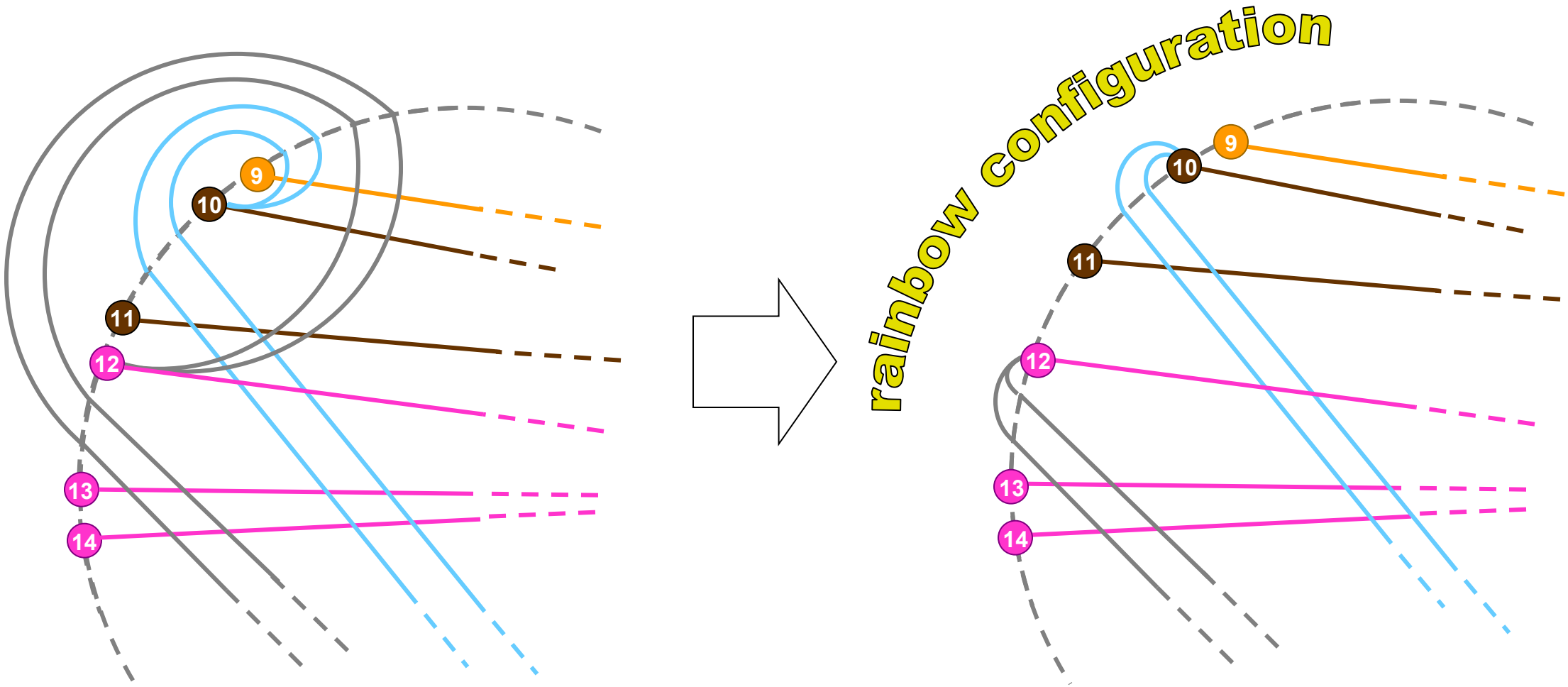
# A two-steps procedure

*Step 1:* Every level of the tree starting from the bottommost is transformed into a *full rainbow configuration* by removing crossings one by one

*Step 2:* Once every level is in full rainbow configuration the remaining crossings are removed one by one until no crossing remains

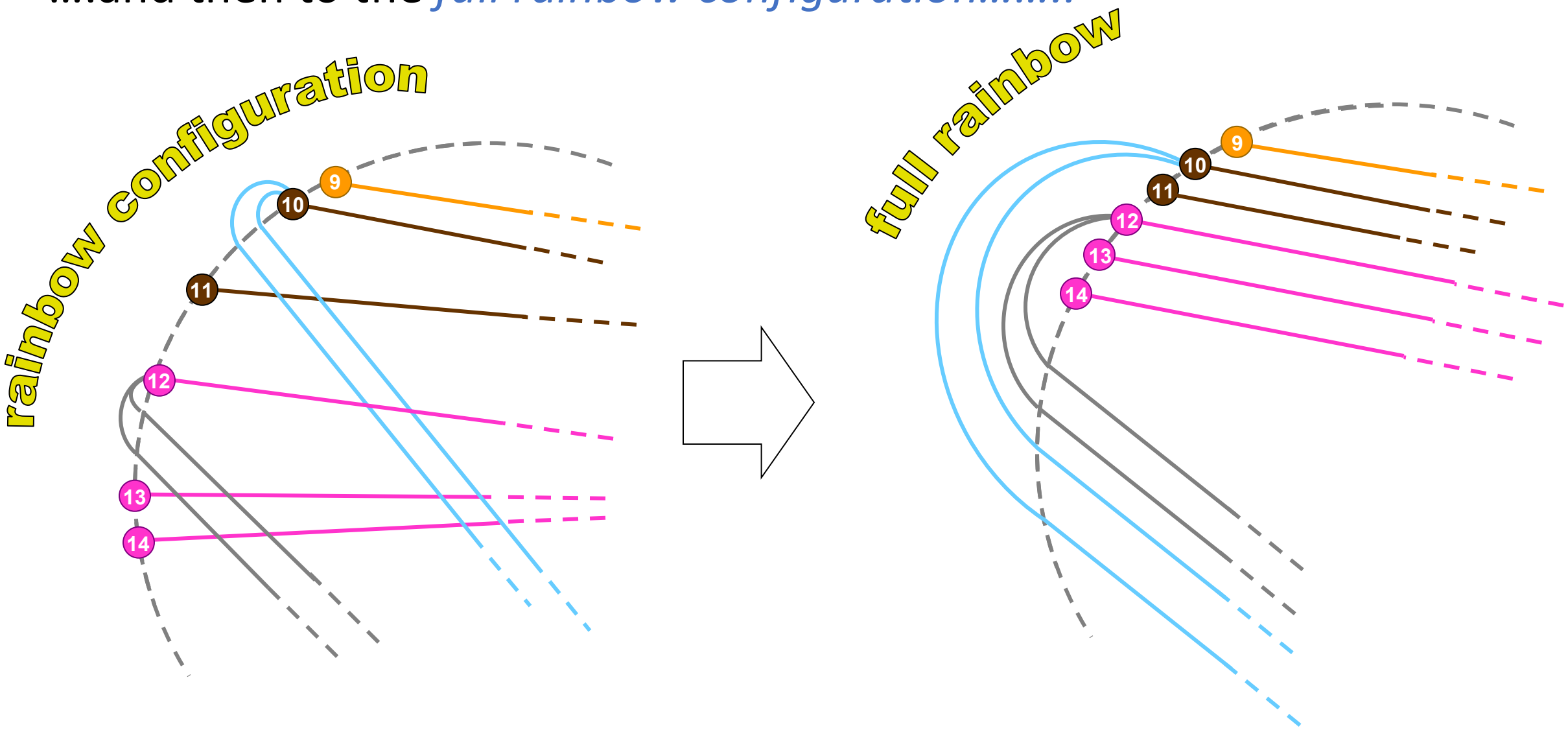
# Step 1: making full rainbows

1.a: a level is modified to its *rainbow configuration……..*

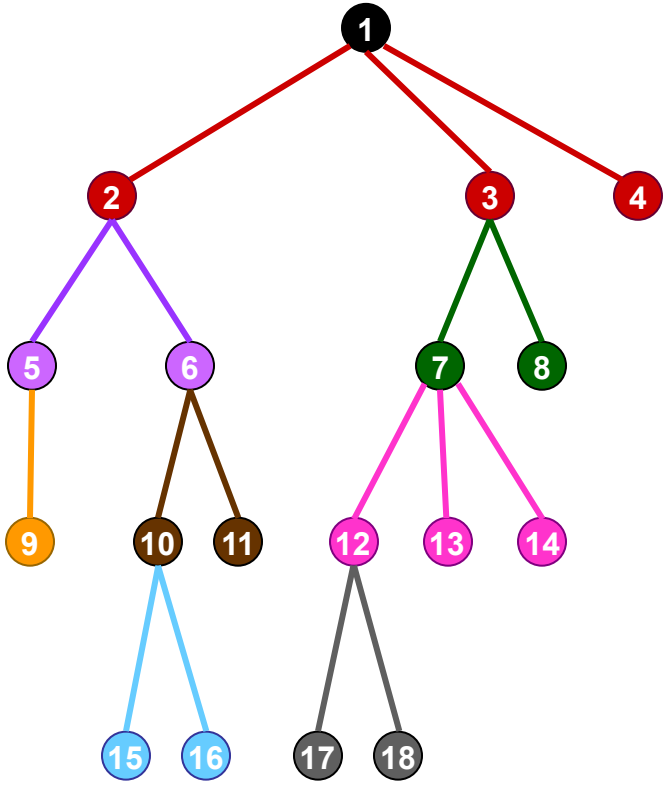


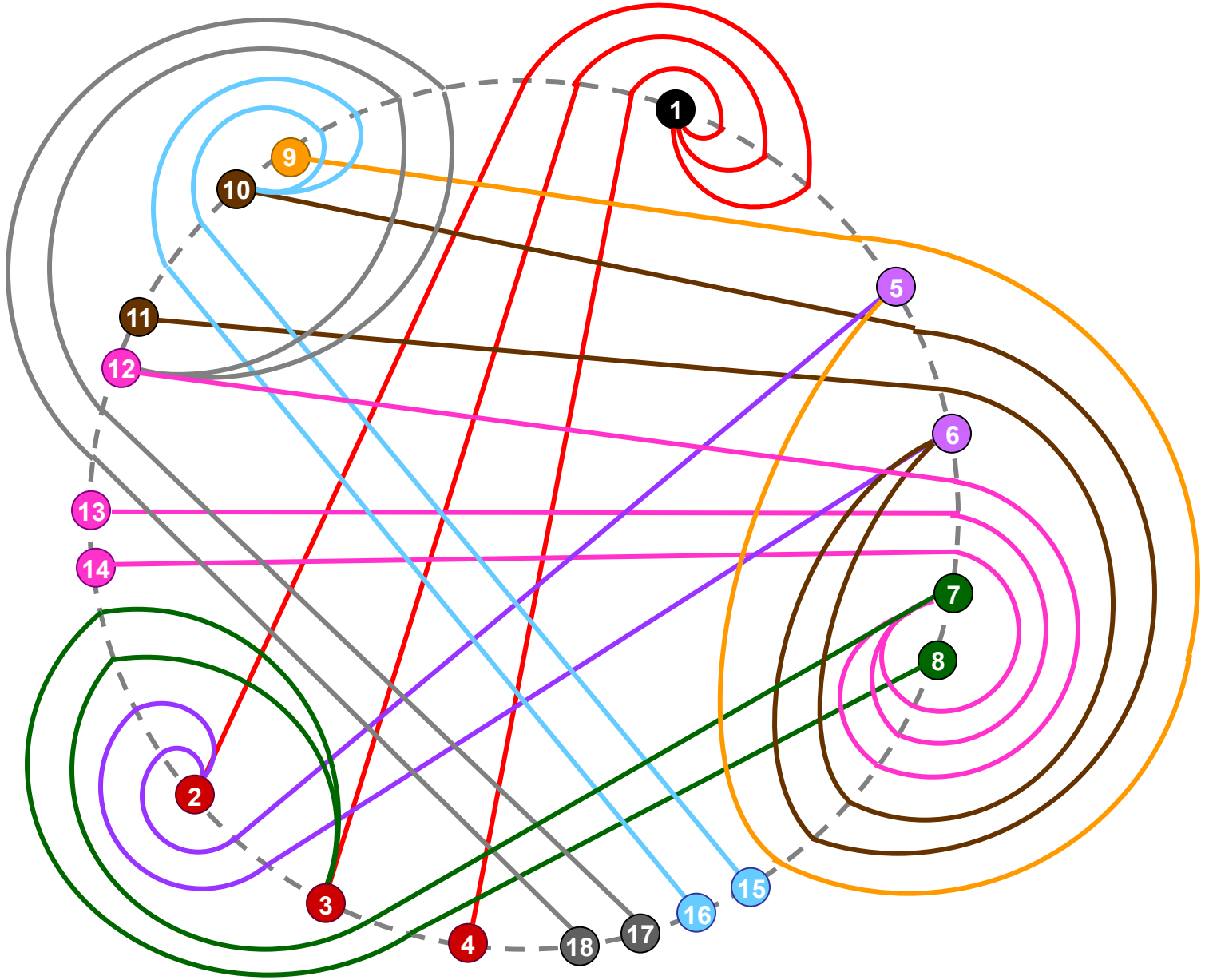
# Step 1: making full rainbows

….and then to the *full rainbow configuration……..*



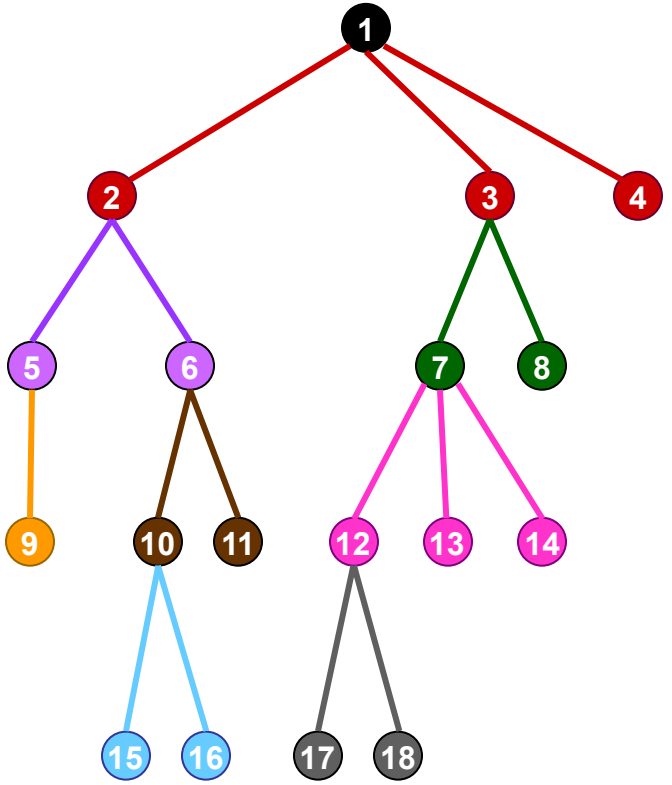
# How to make rainbows

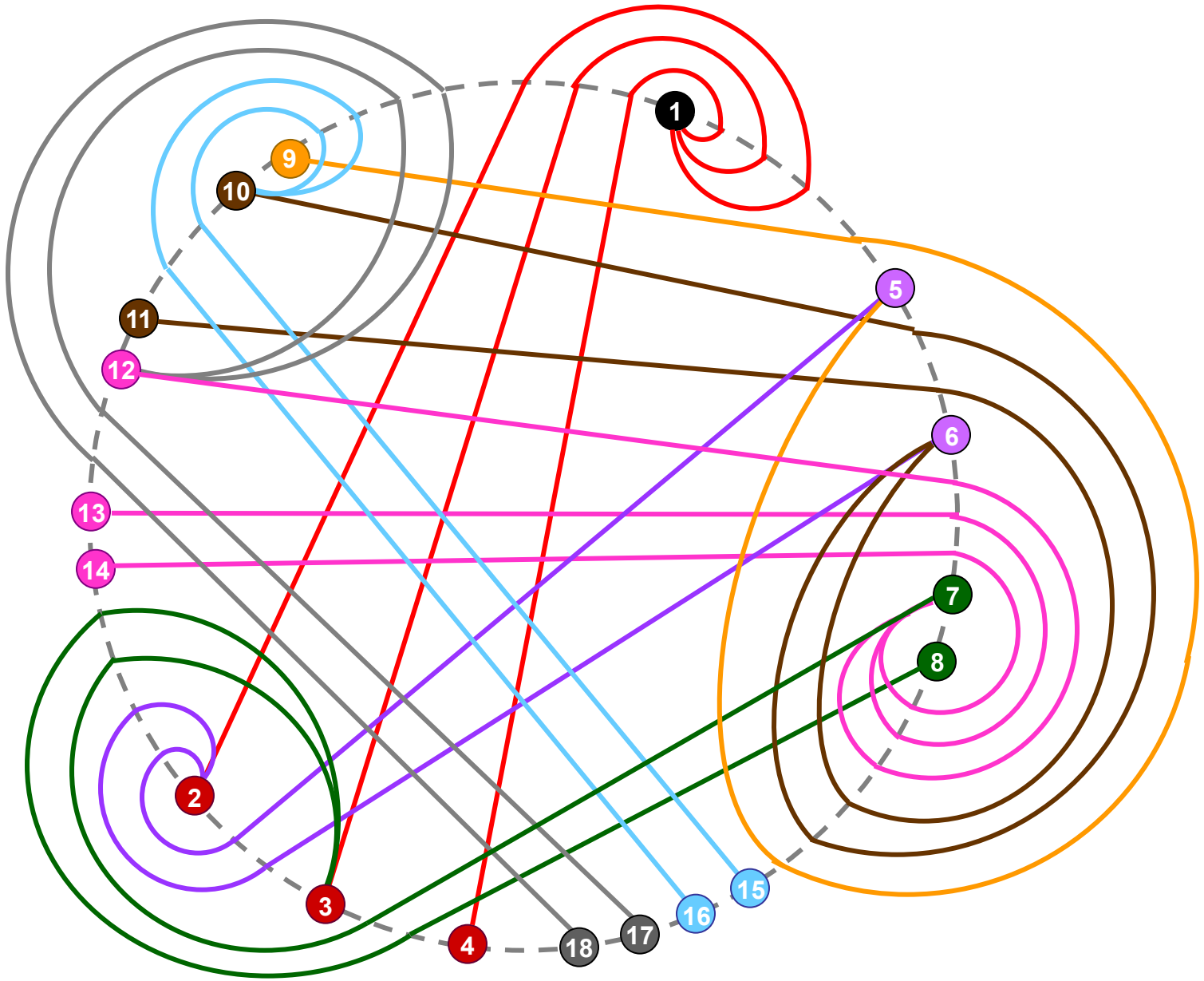


# How to make rainbows

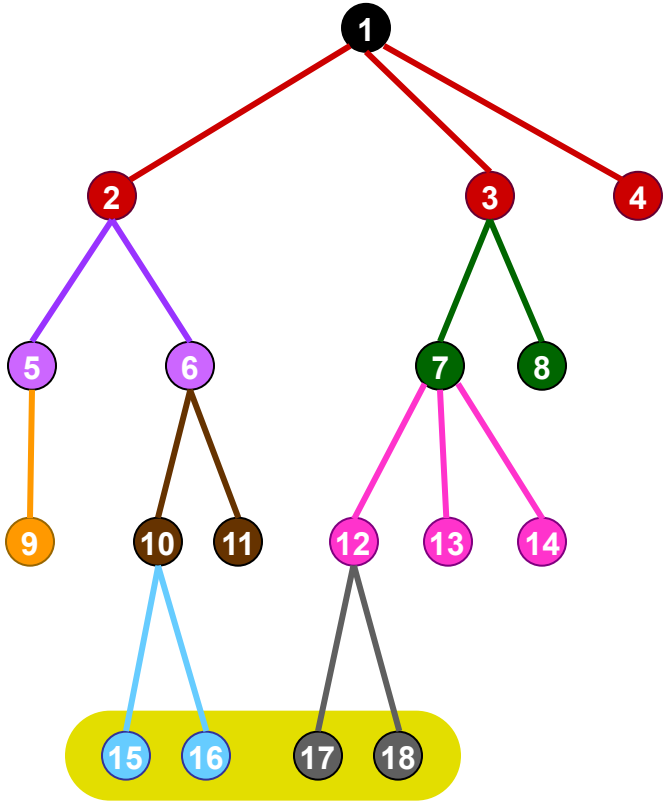
**Inductive hypothesis:** for level k, all levels larger than k are in full rainbow

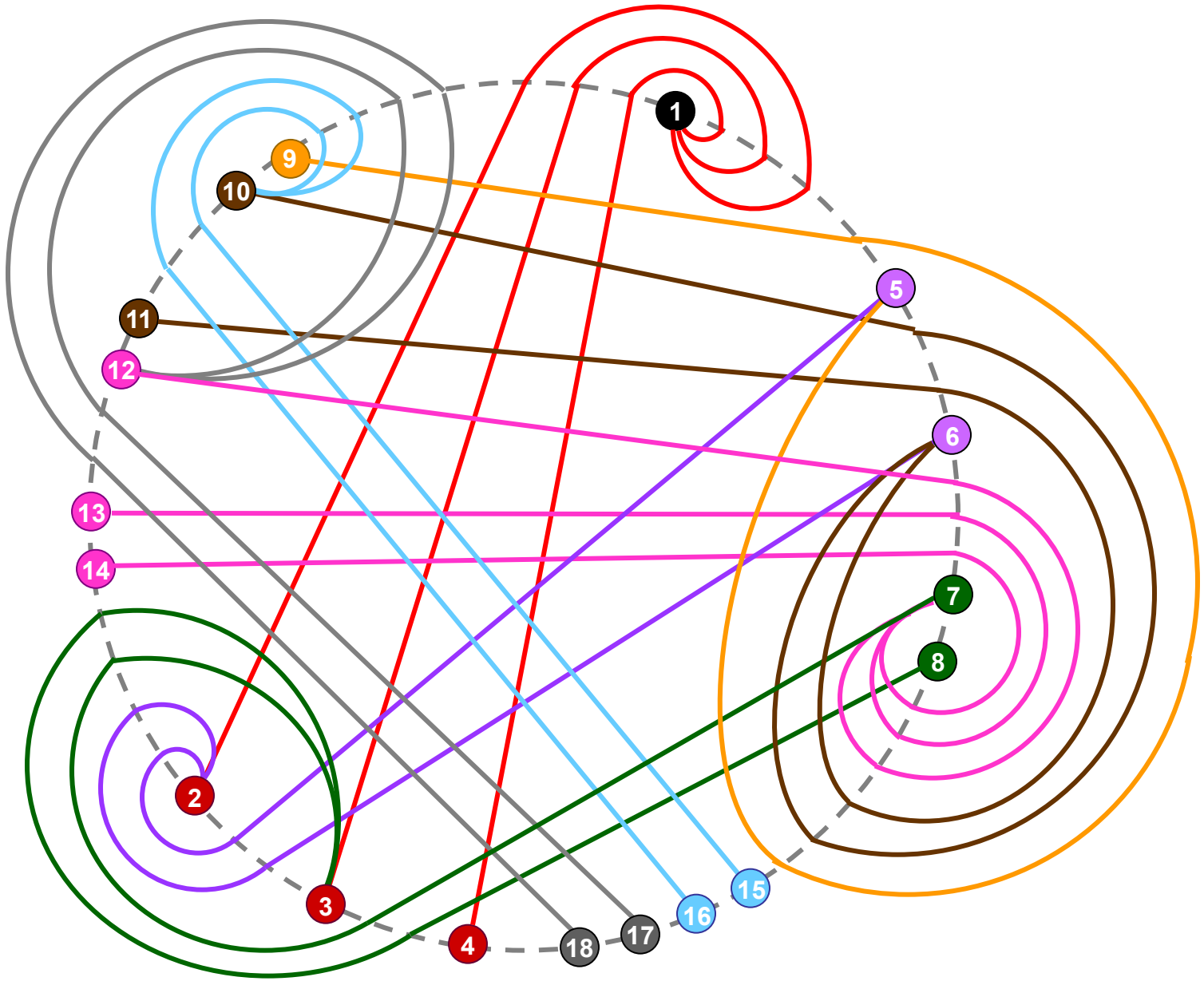


# How to make rainbows

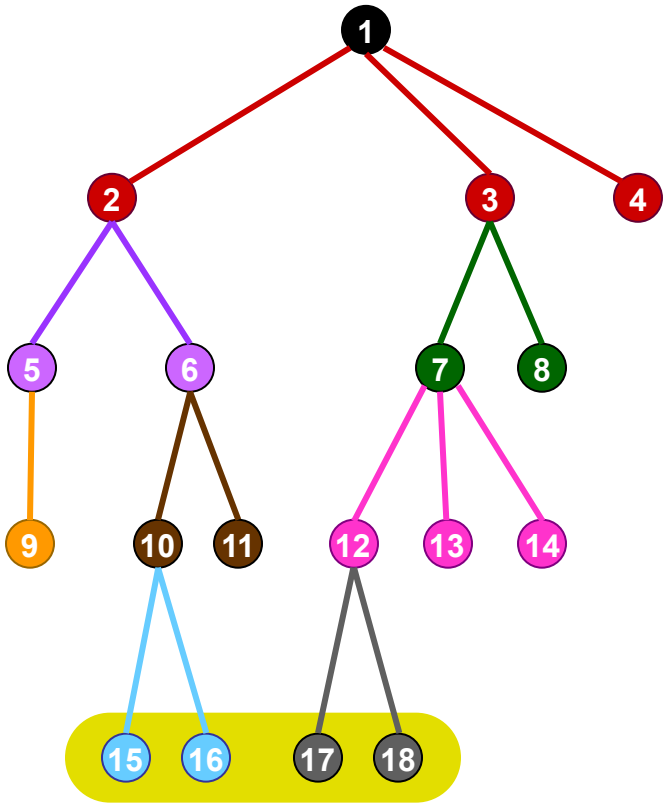
**Inductive hypothesis:** for level k, all levels larger than k are in full rainbow

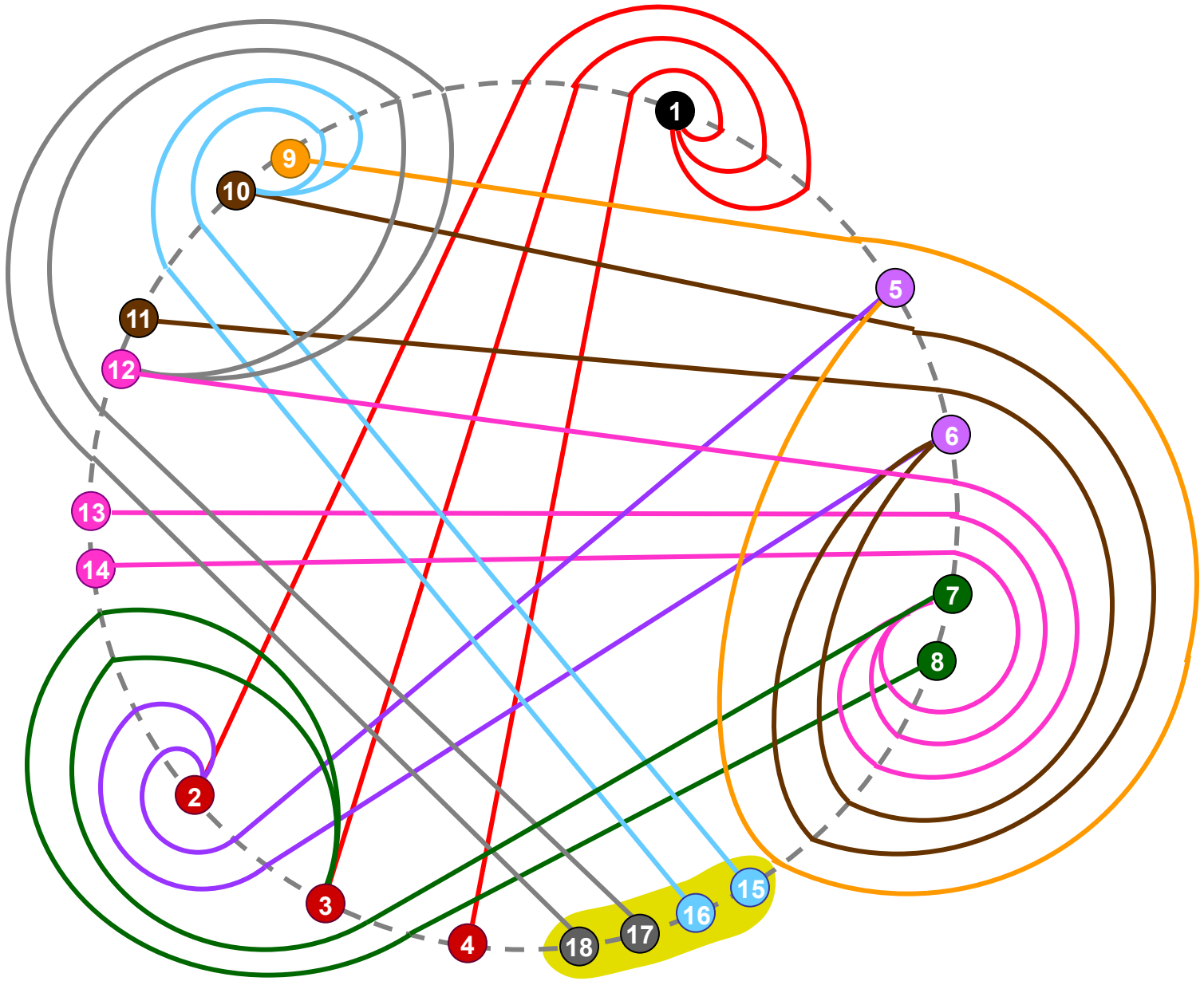


# How to make rainbows

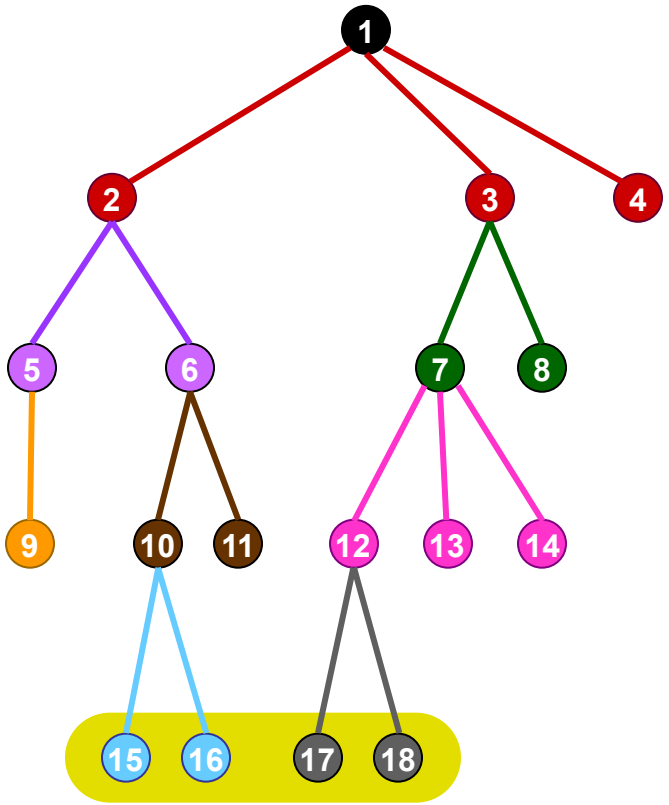
**Inductive hypothesis:** for level k, all levels larger than k are in full rainbow

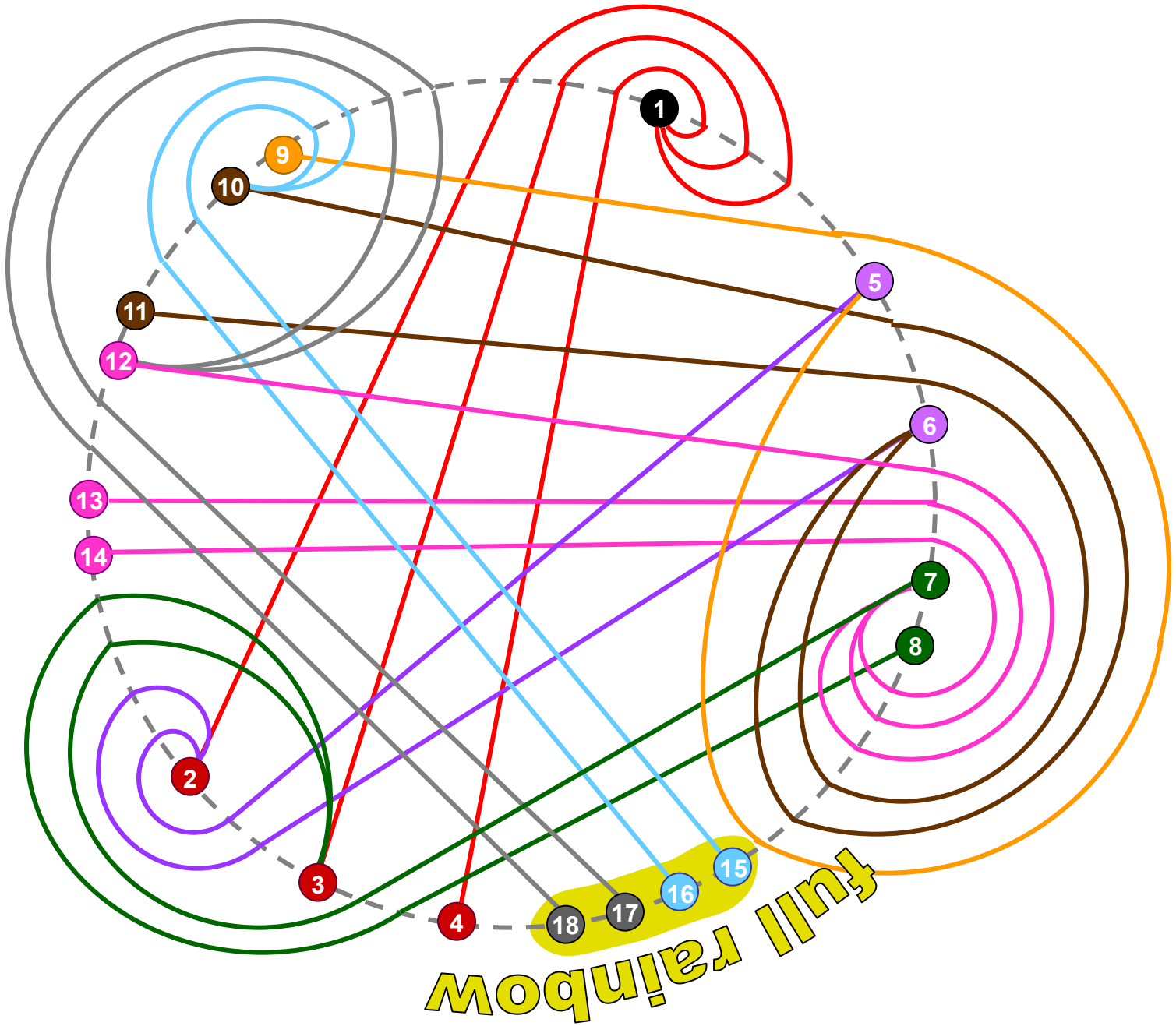


# How to make rainbows

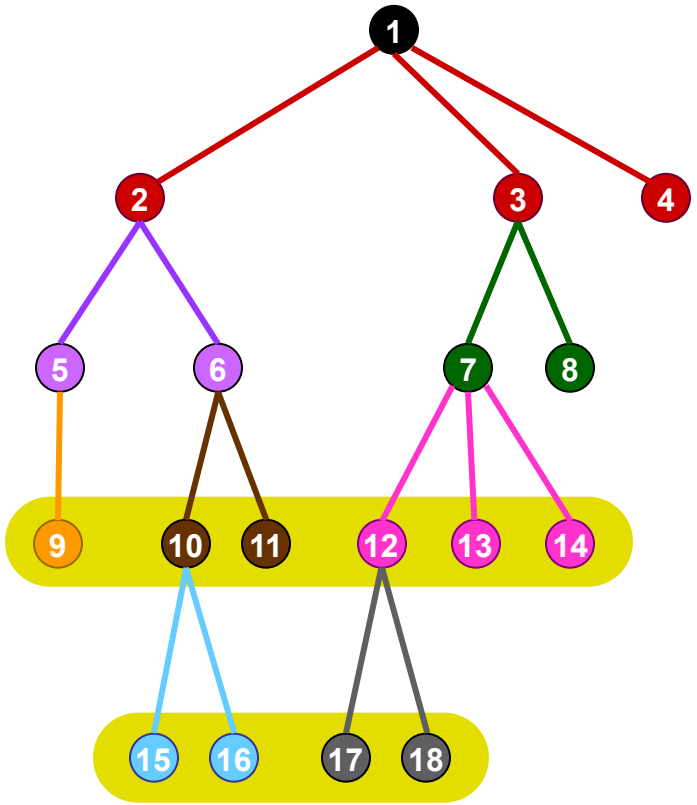
**Inductive hypothesis:** for level k, all levels larger than k are in full rainbow

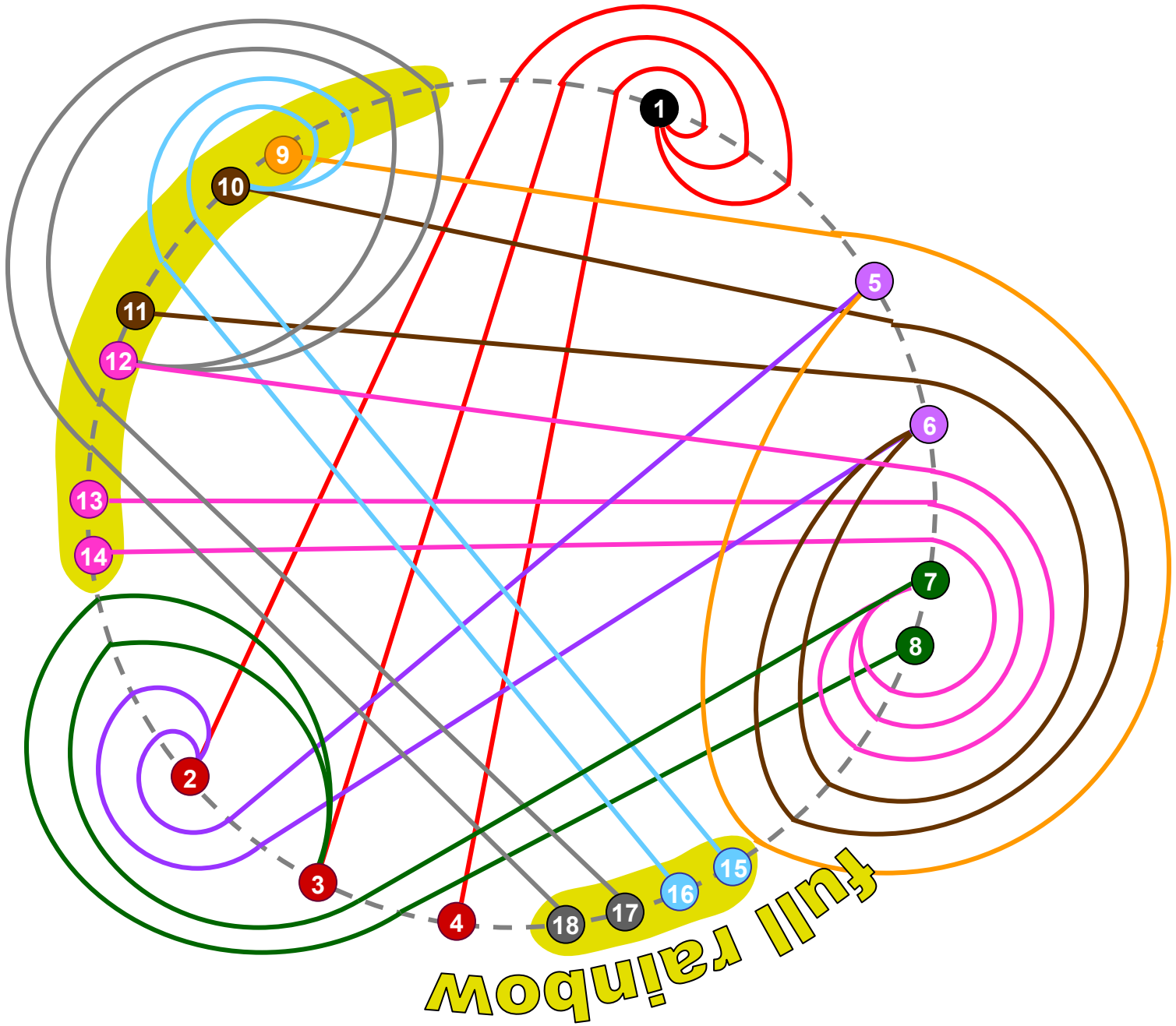


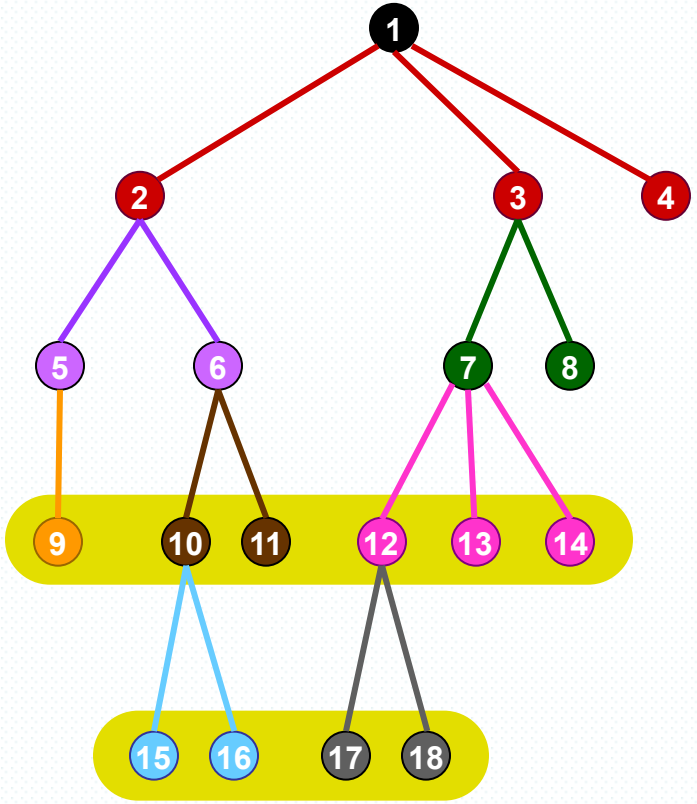
# How to make rainbows

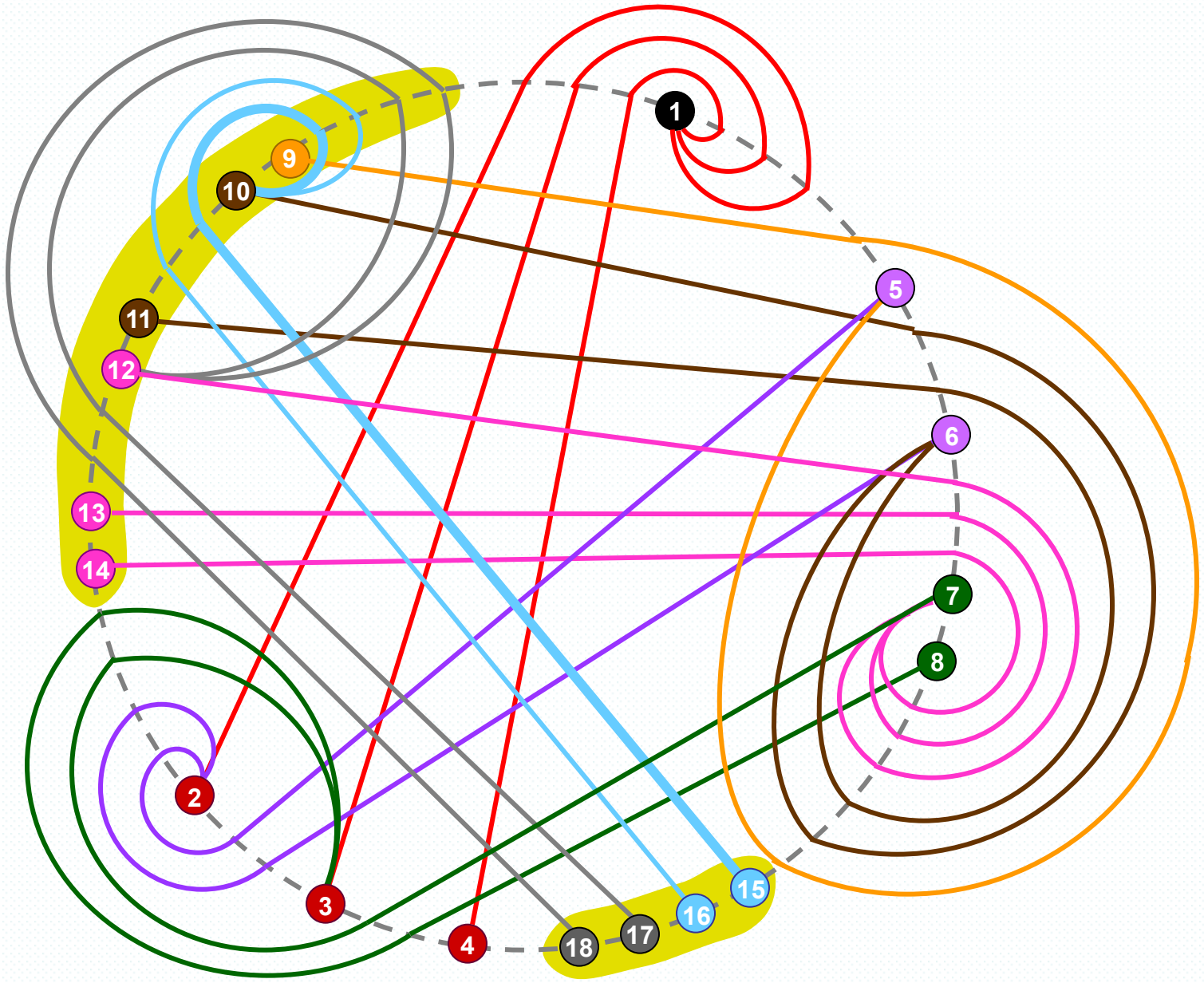
**Inductive hypothesis:** for level k, all levels larger than k are in full rainbow



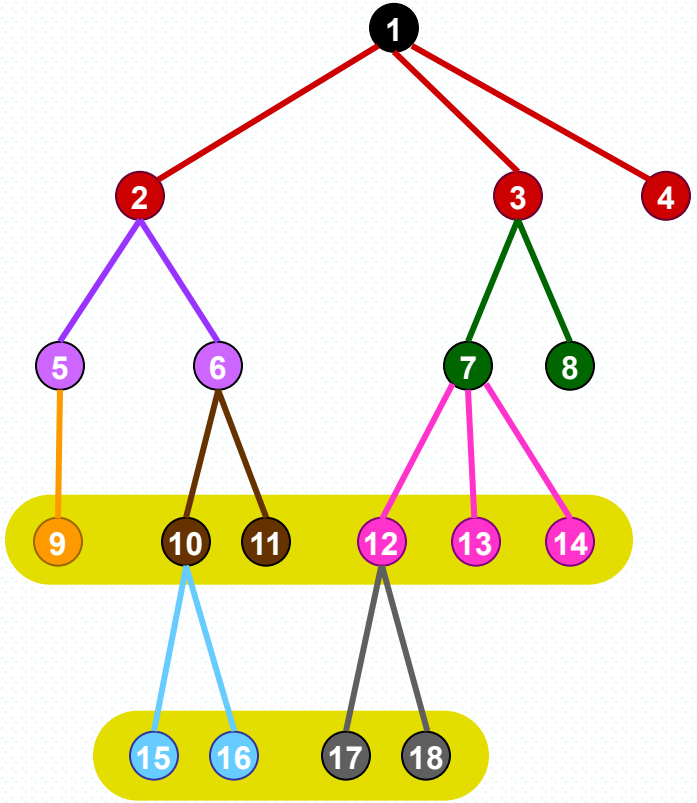



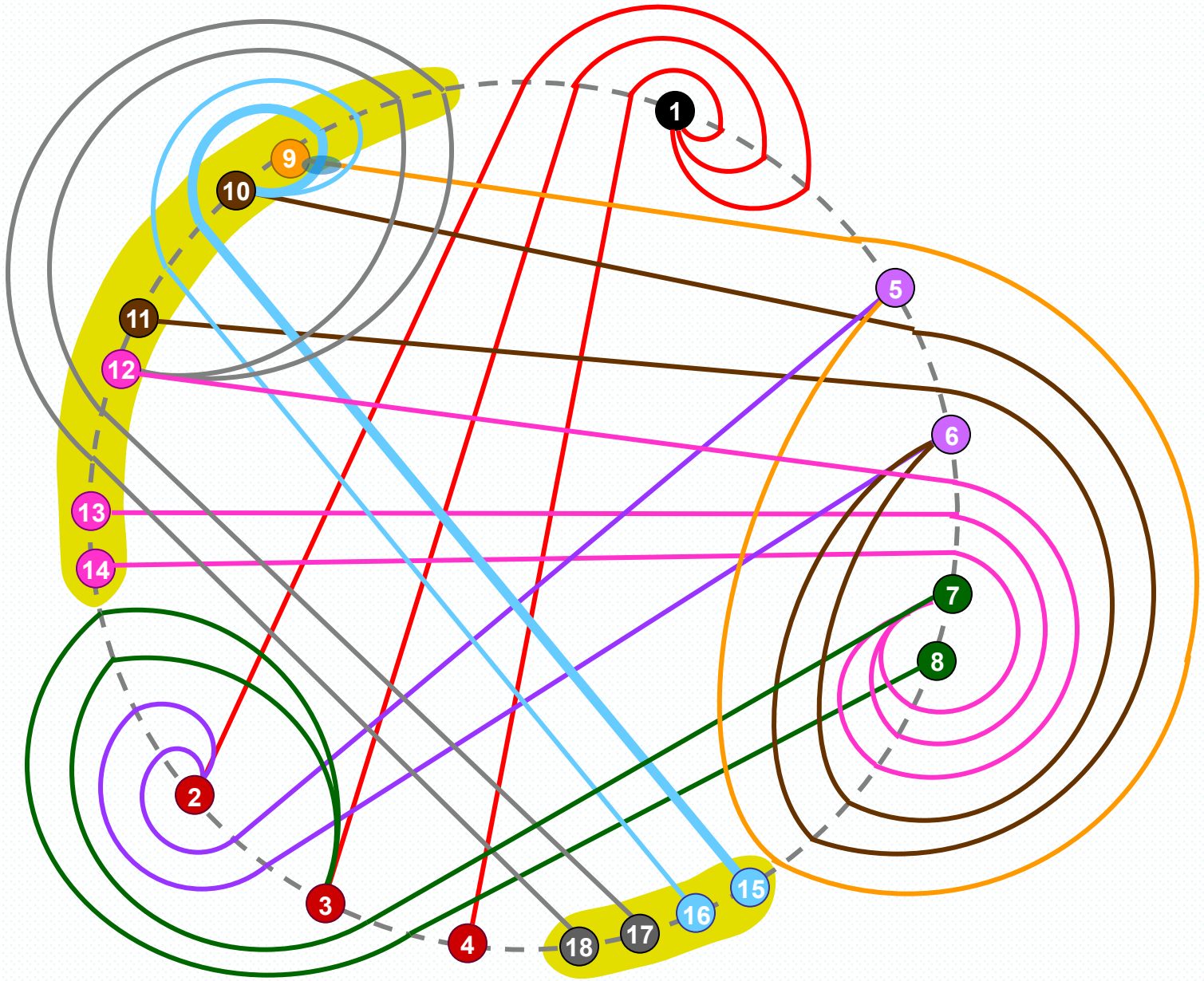
# How to make rainbows



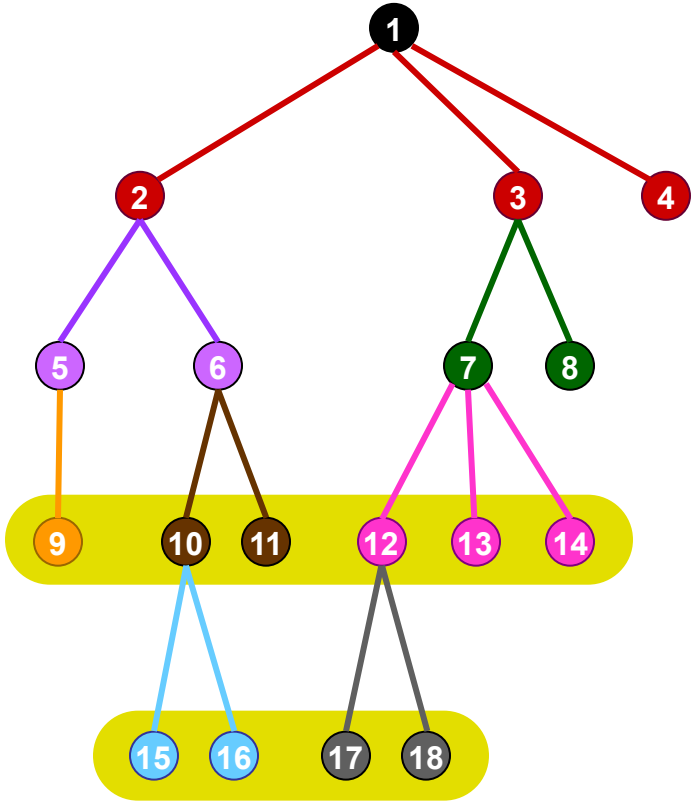



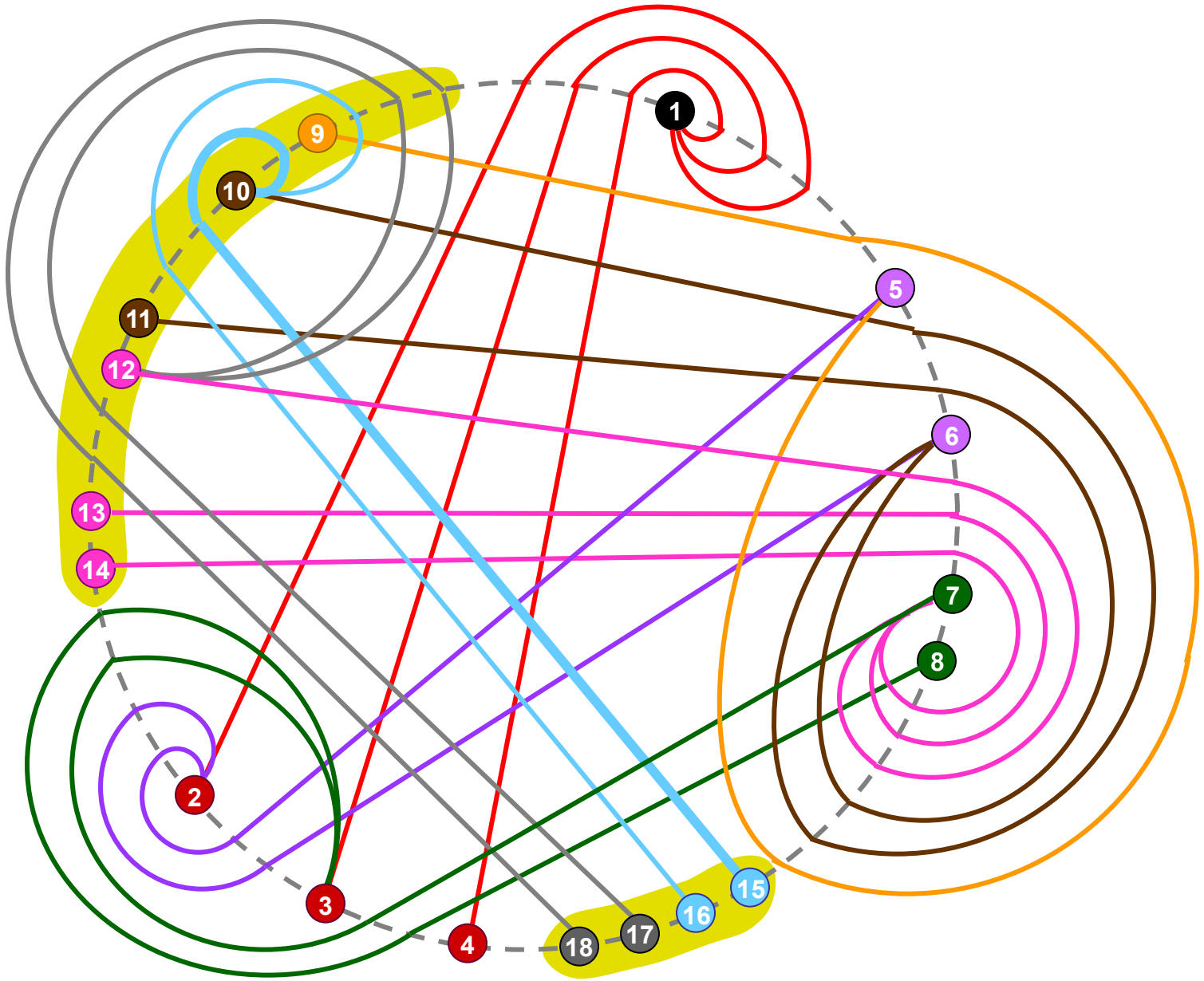
# How to make rainbows



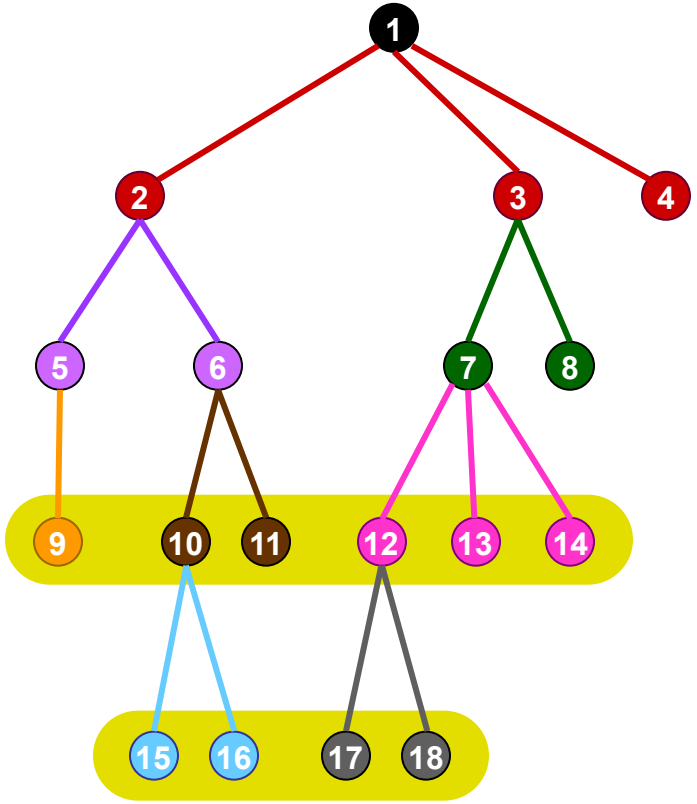



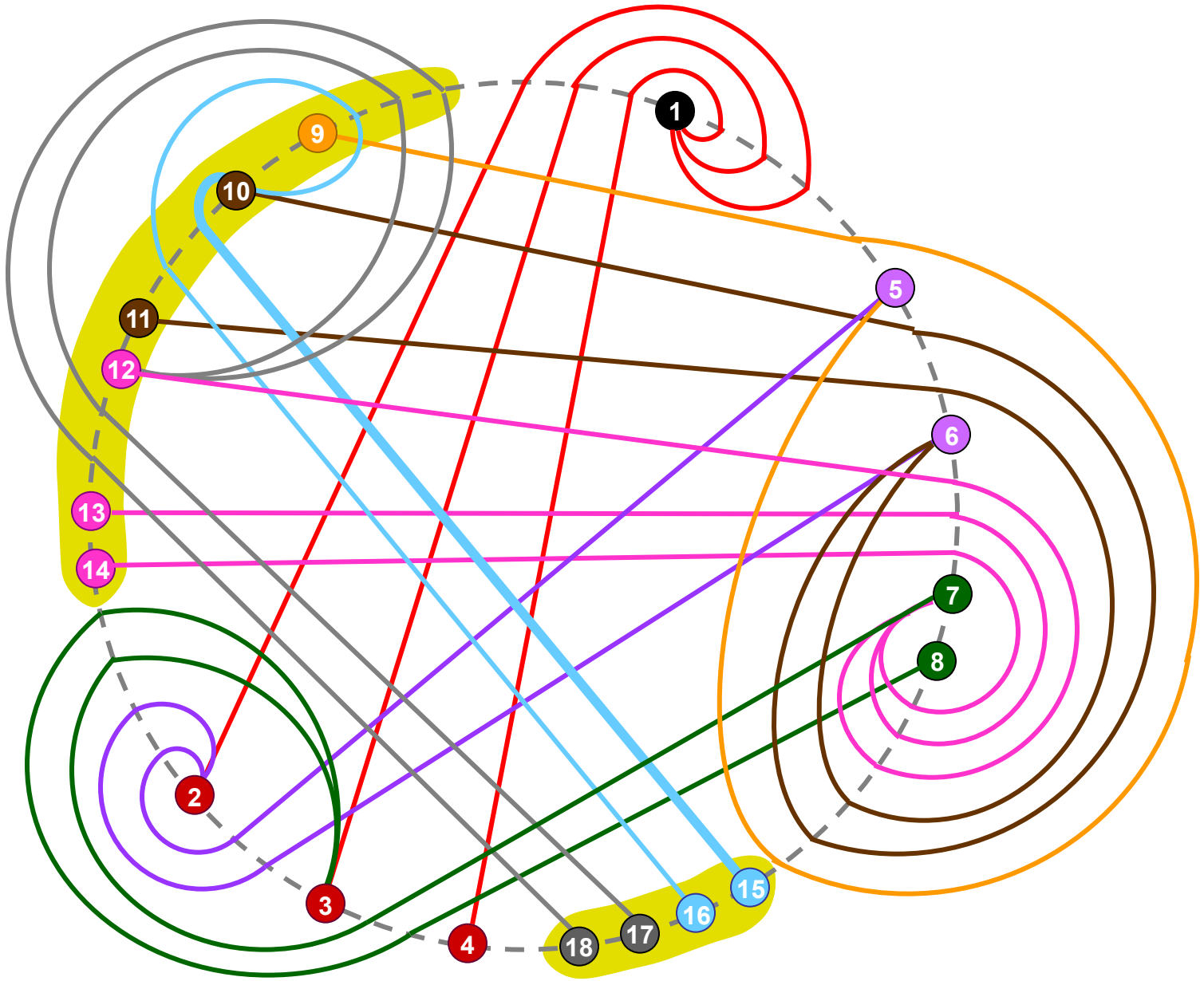
# How to make rainbows



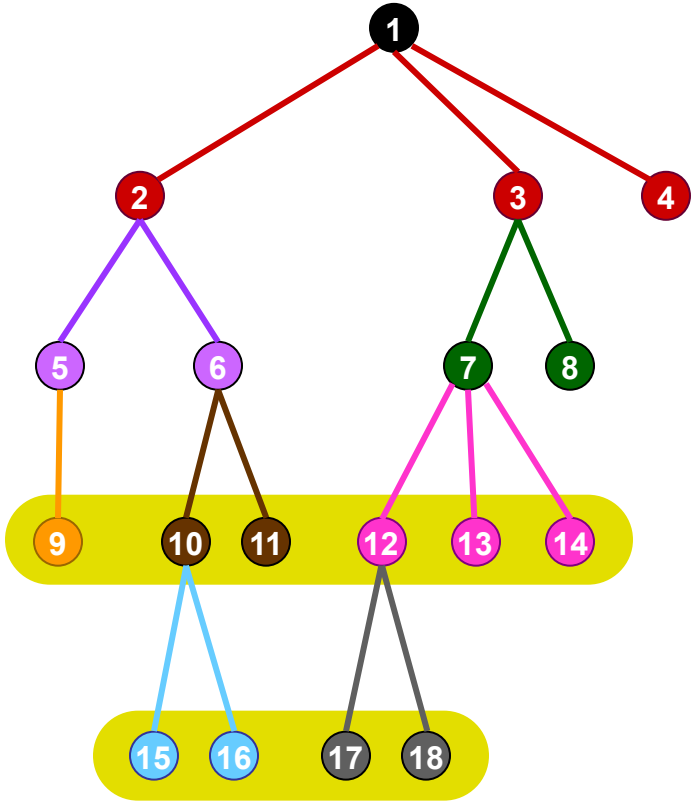



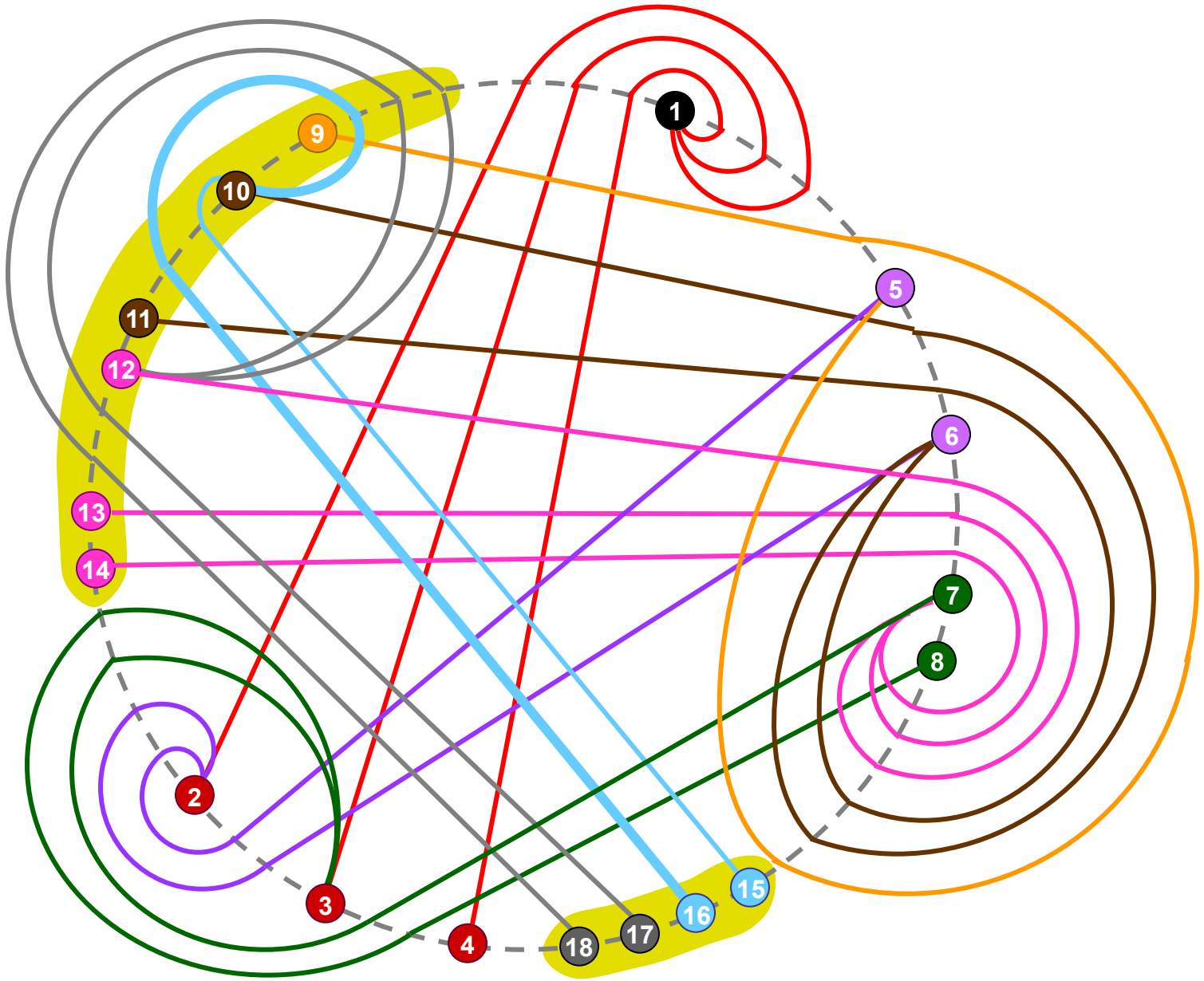
# How to make rainbows



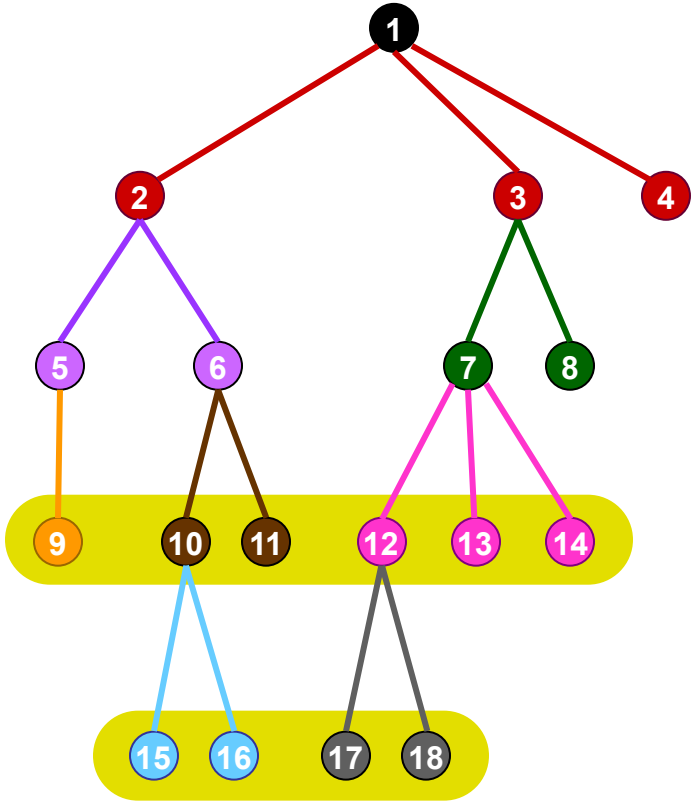



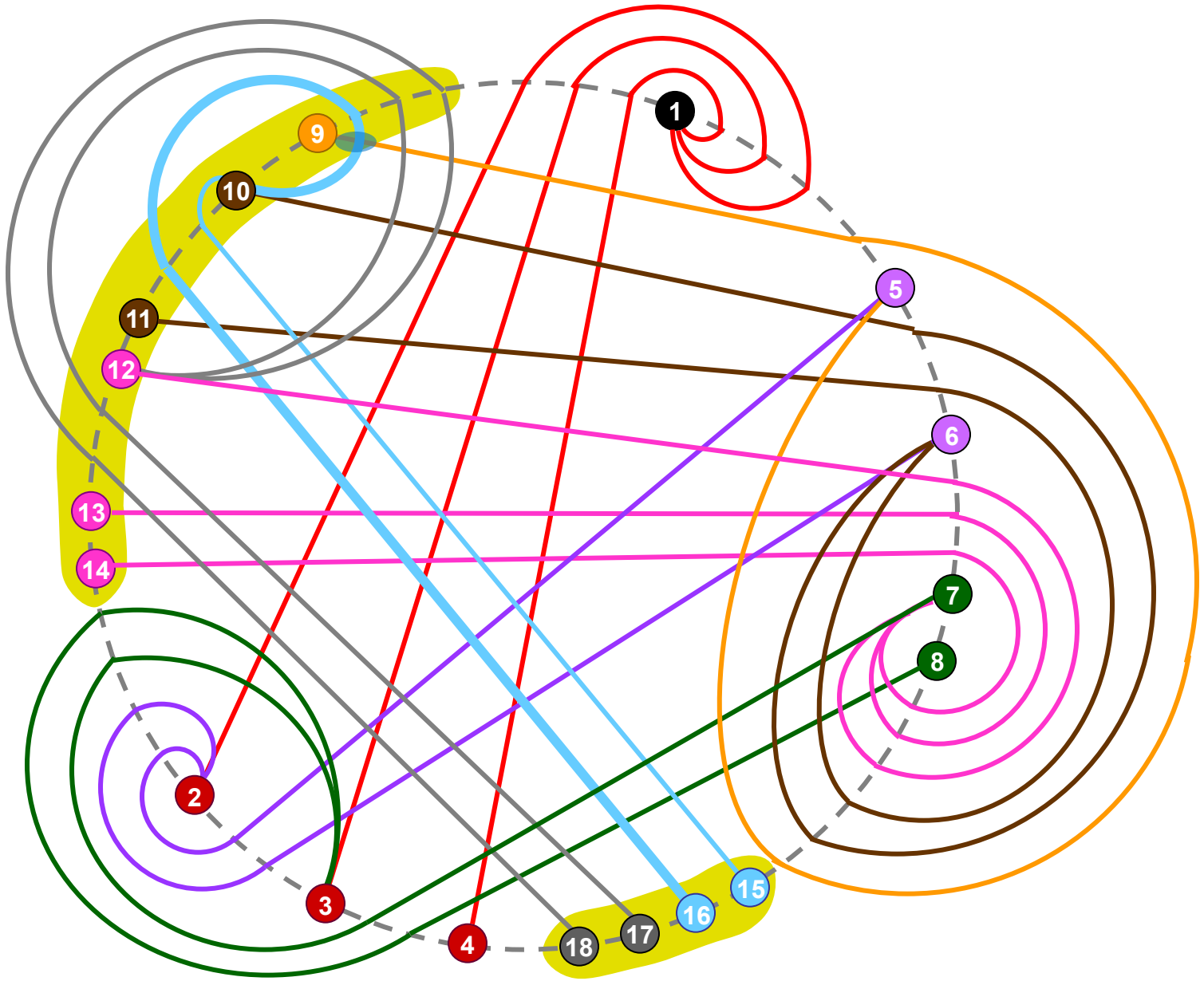
# How to make rainbows



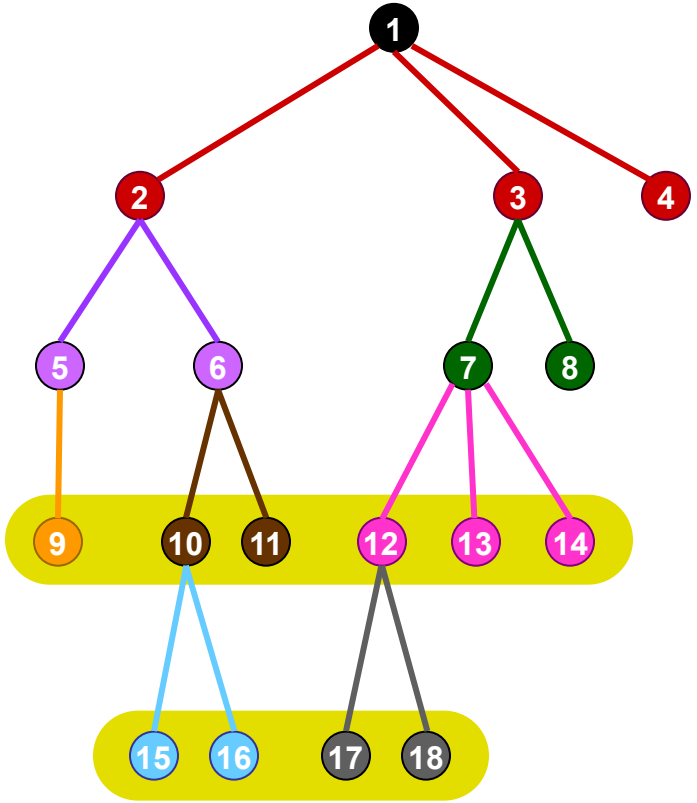



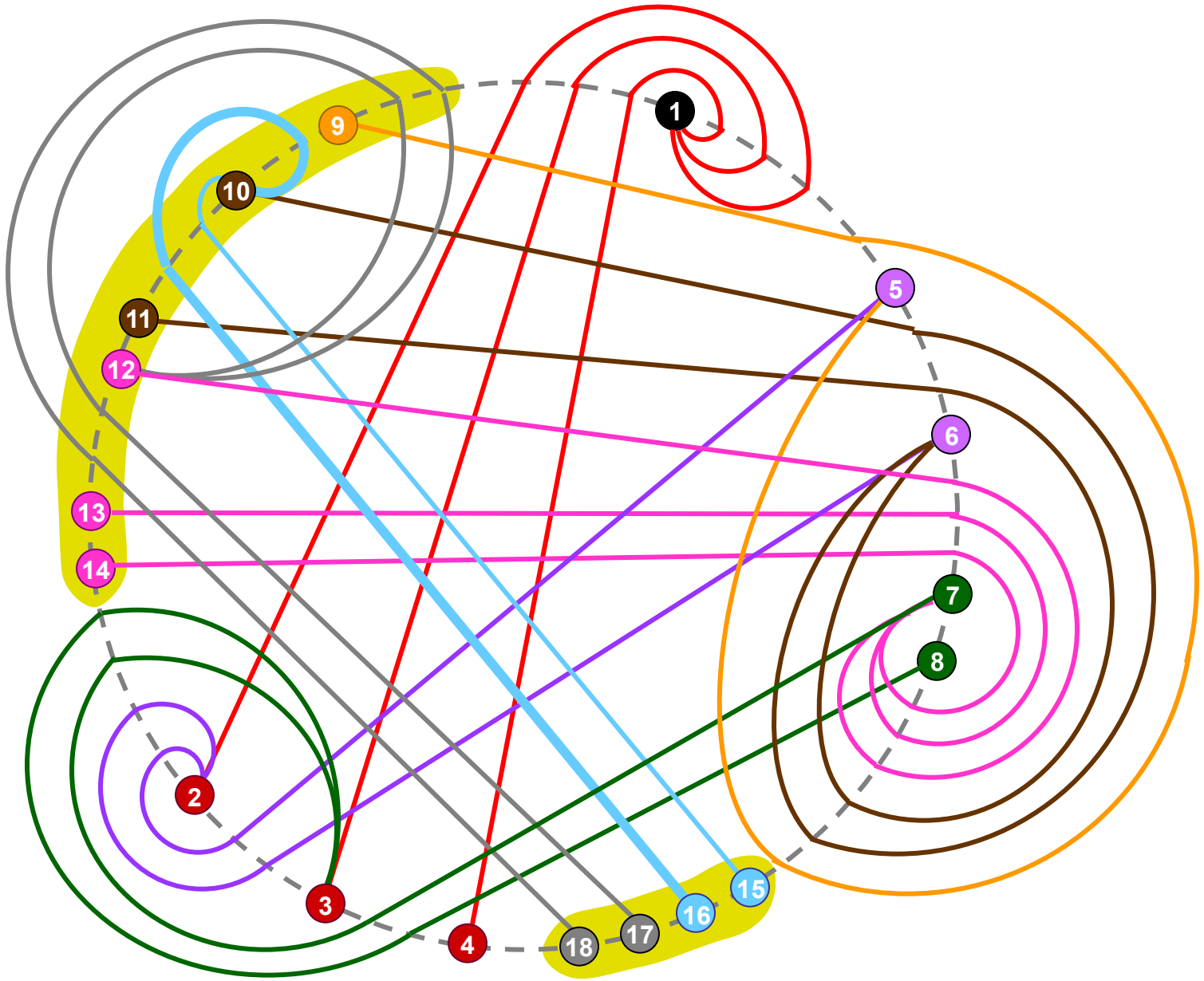
# How to make rainbows



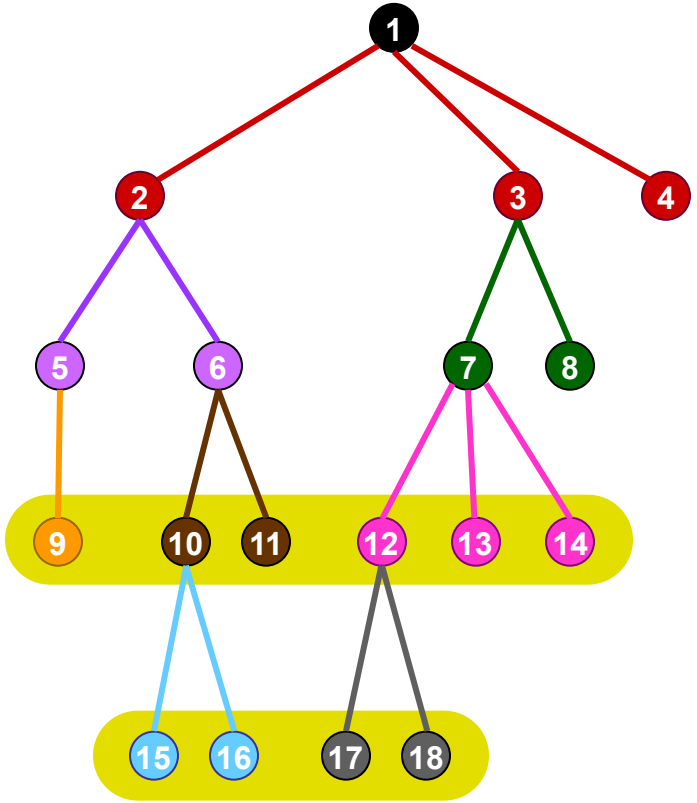



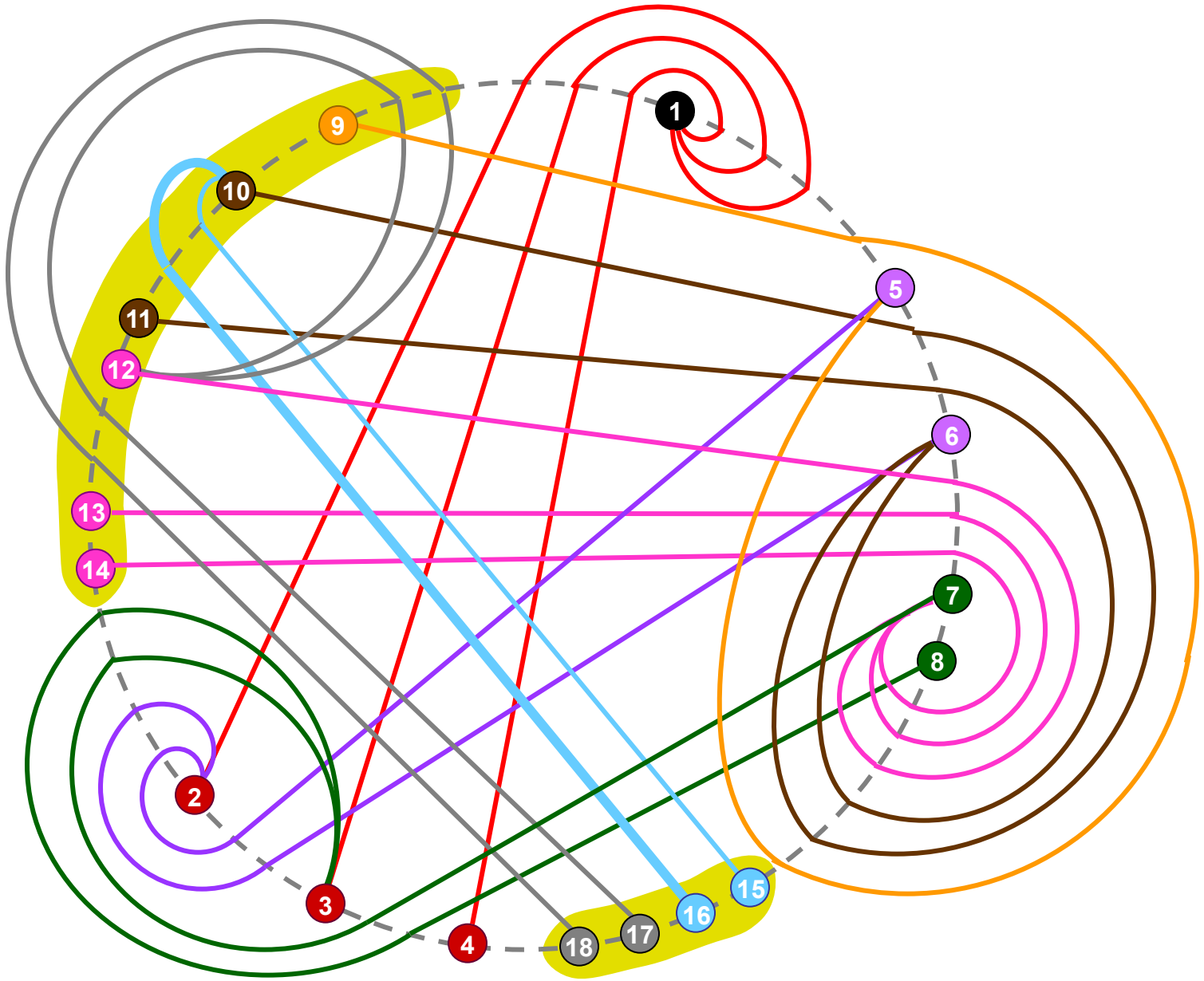
# How to make rainbows



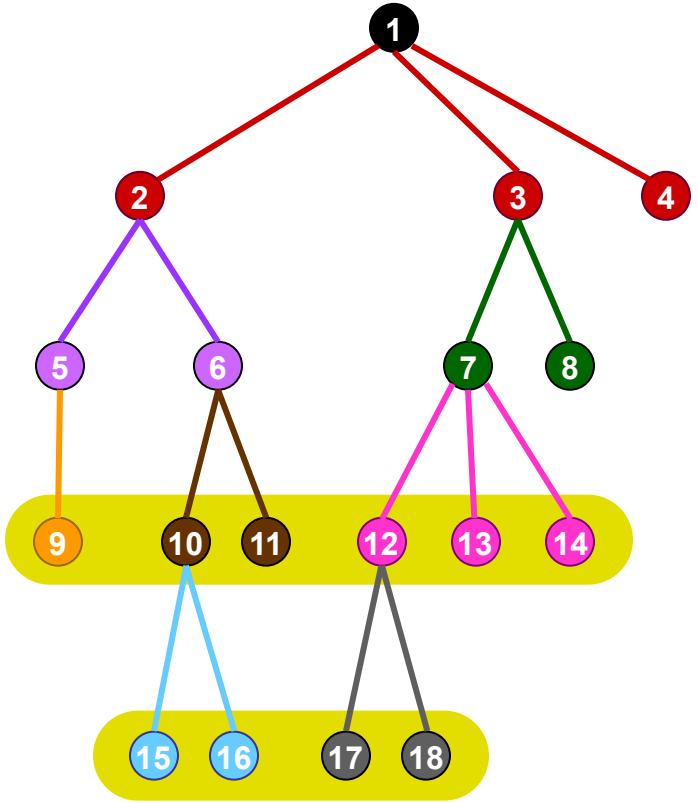



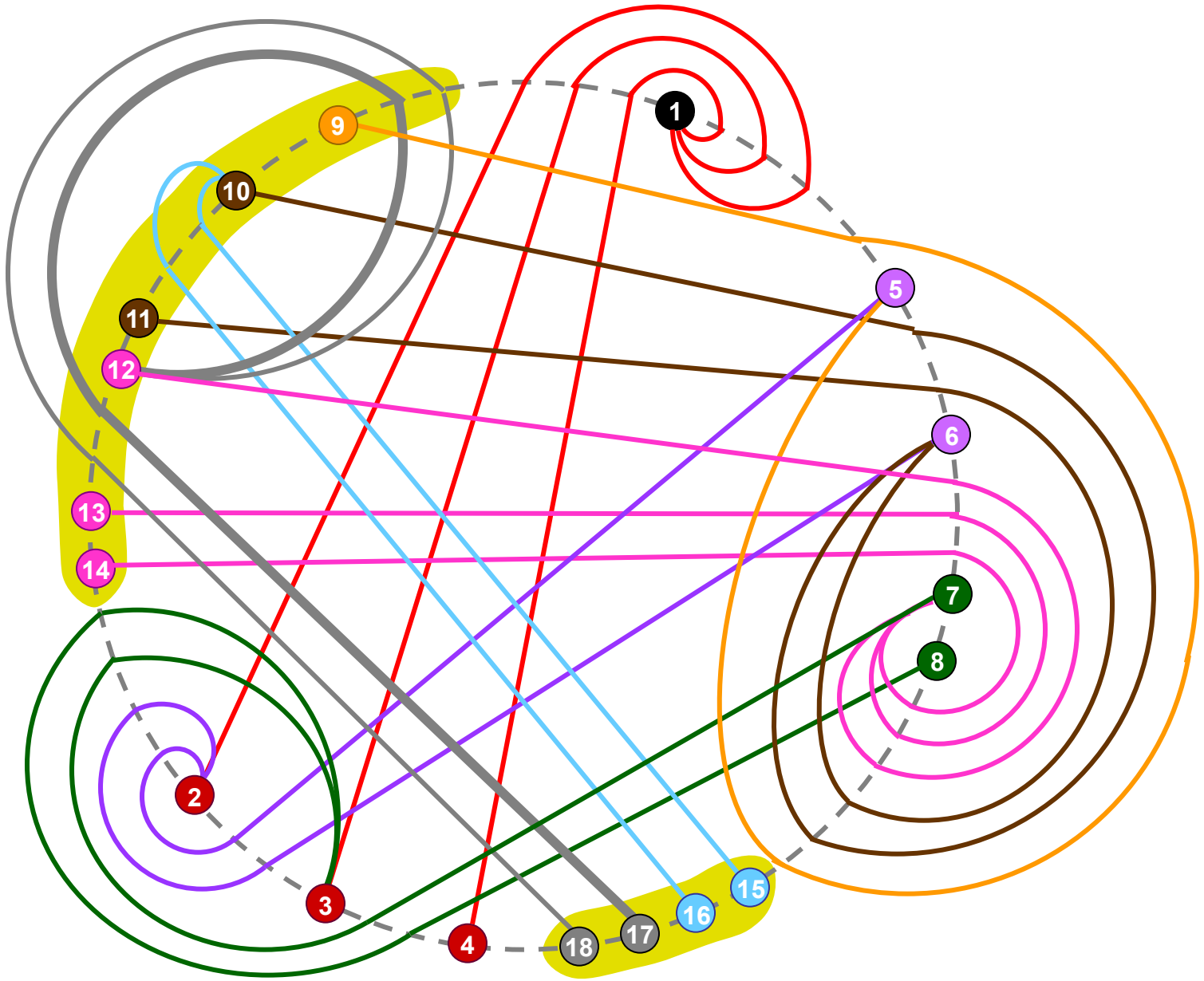
# How to make rainbows



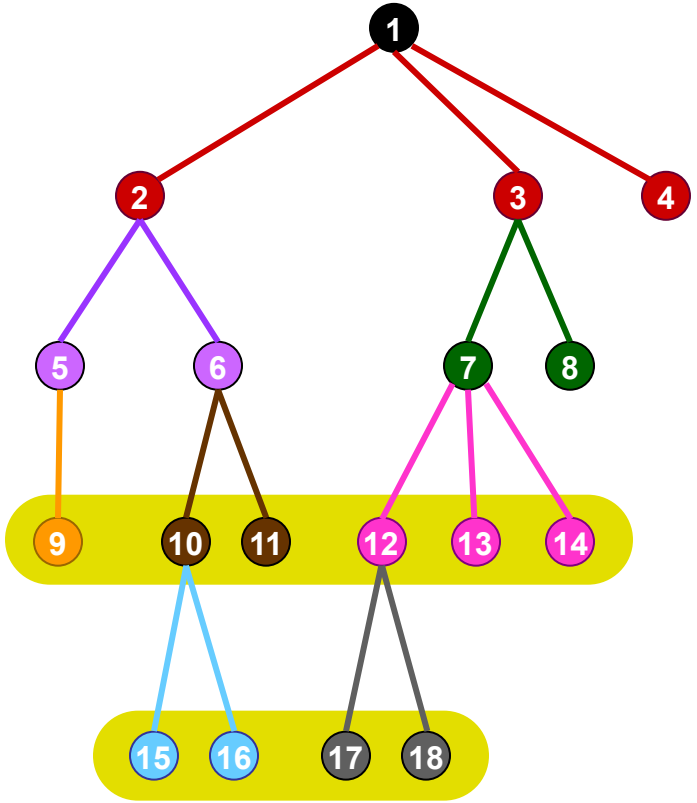



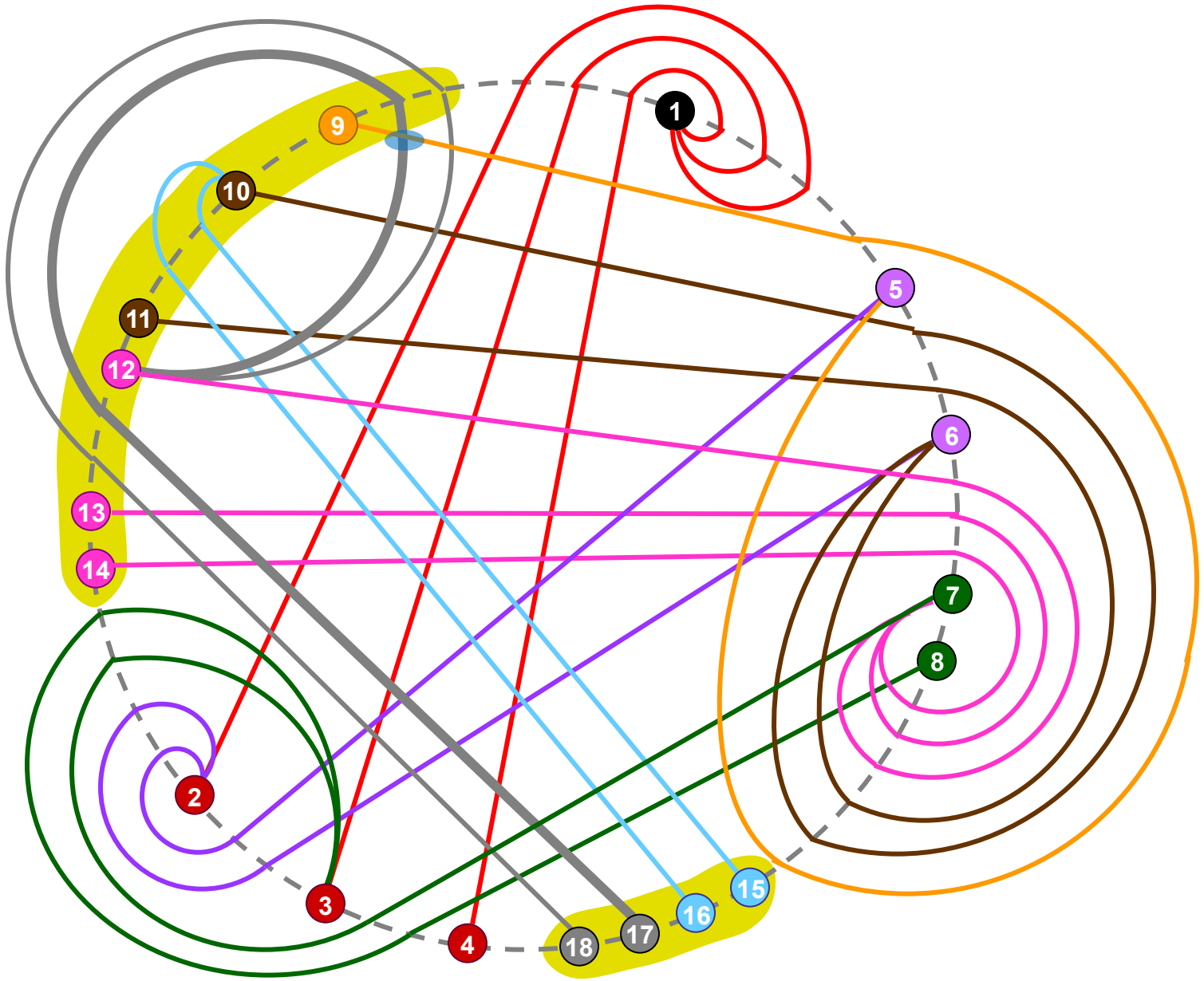
# How to make rainbows



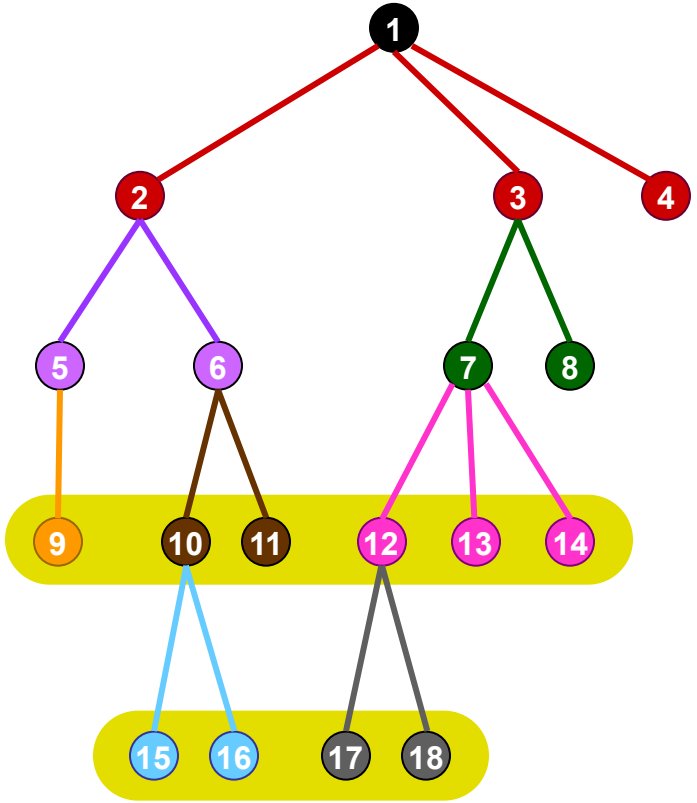



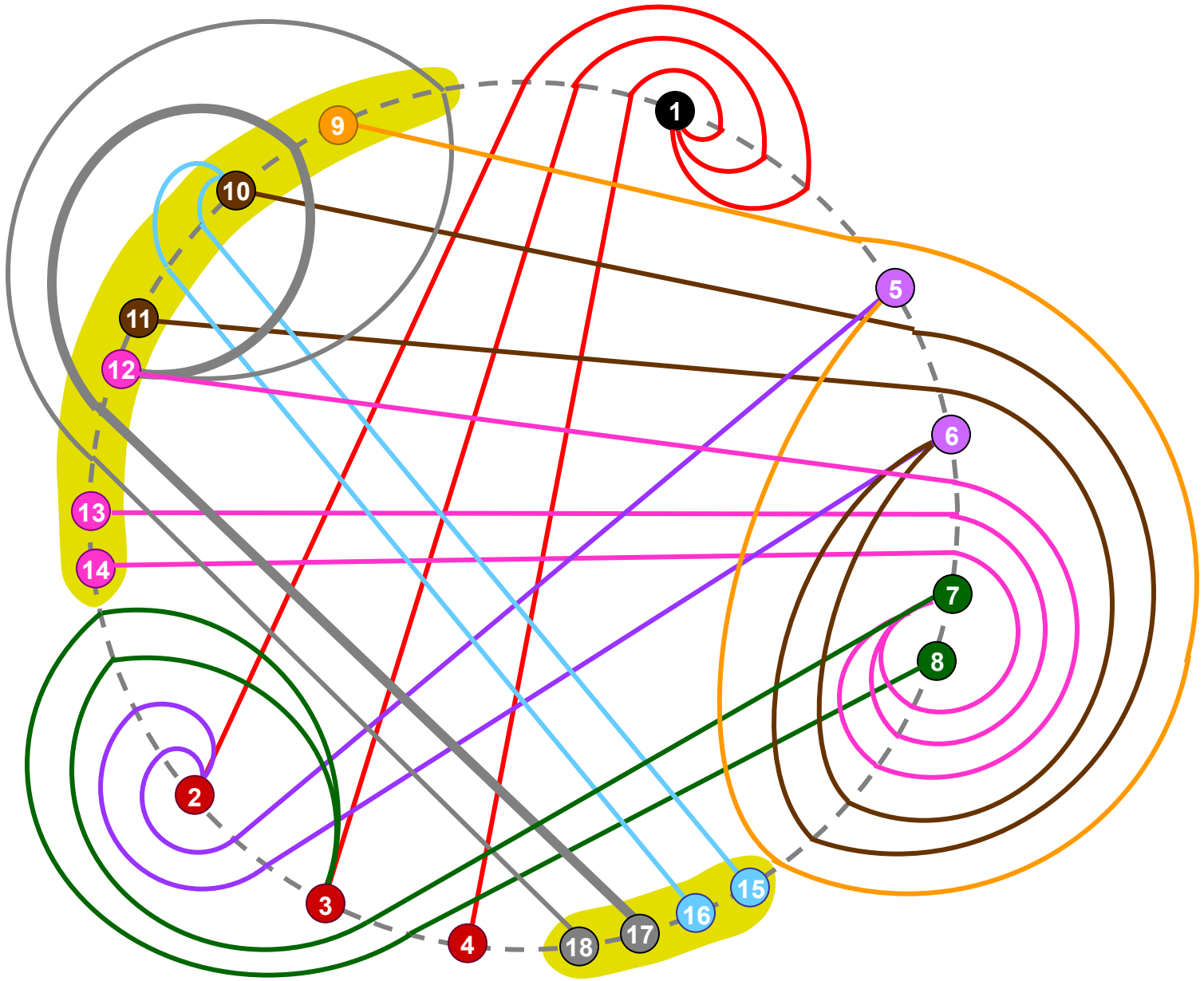
# How to make rainbows



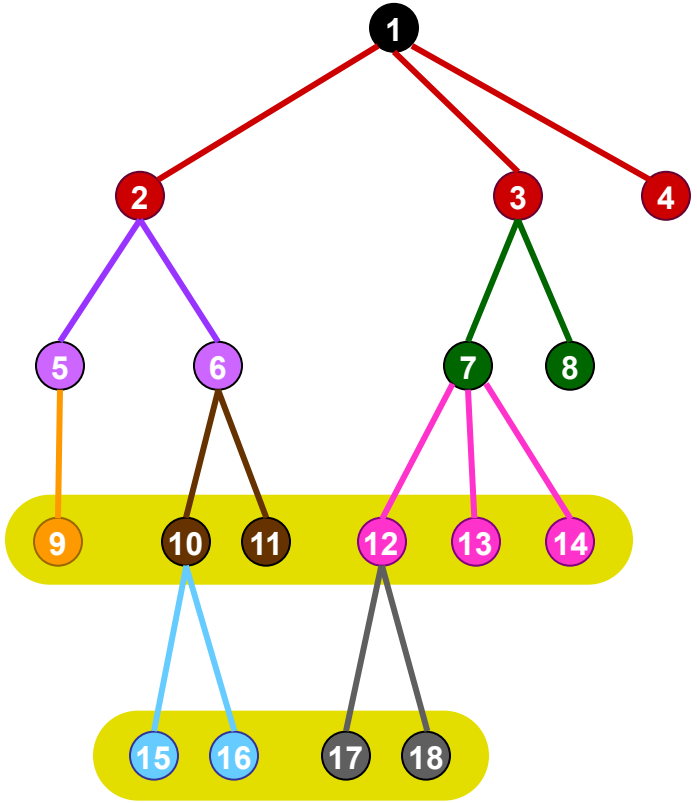



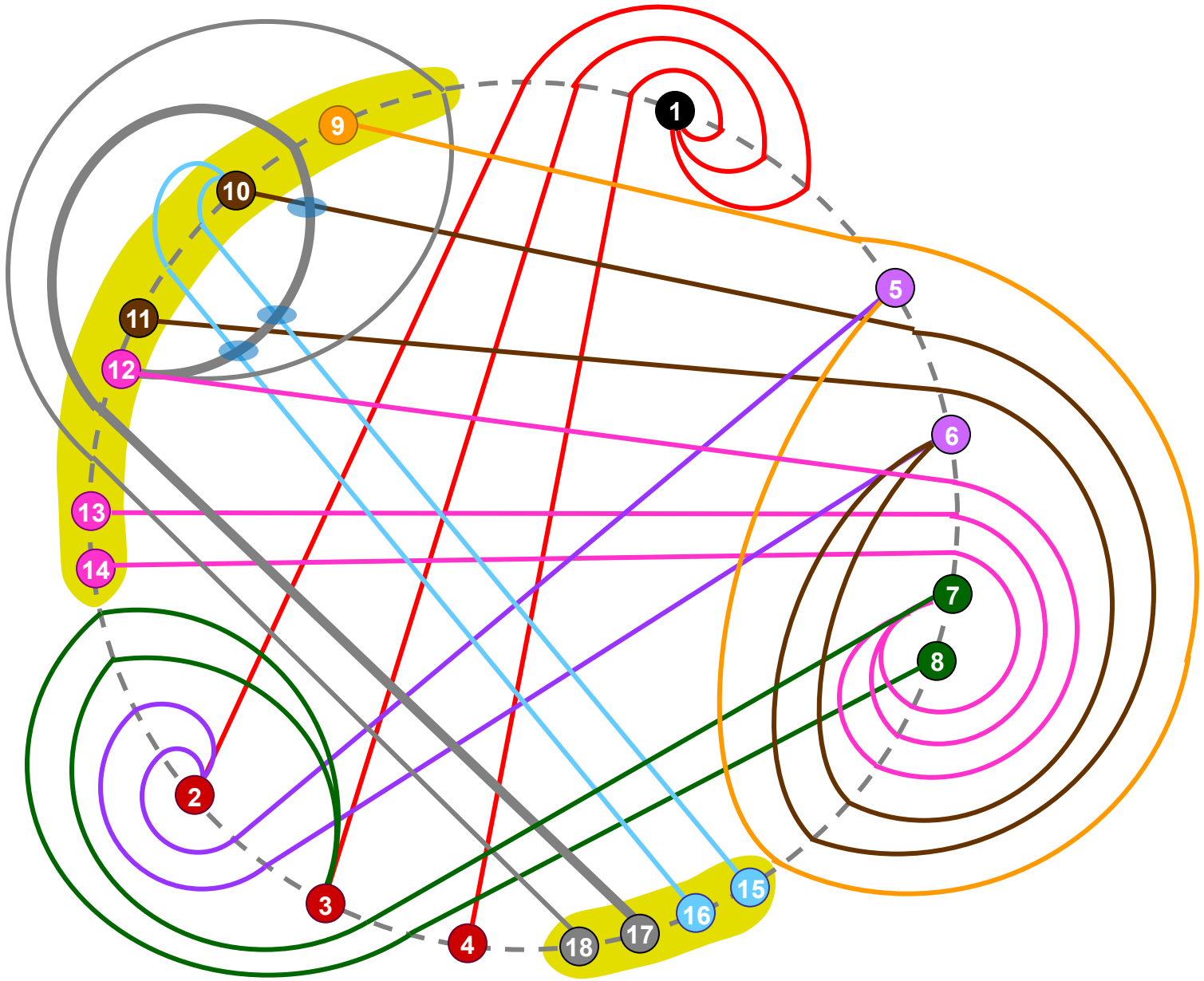
# How to make rainbows



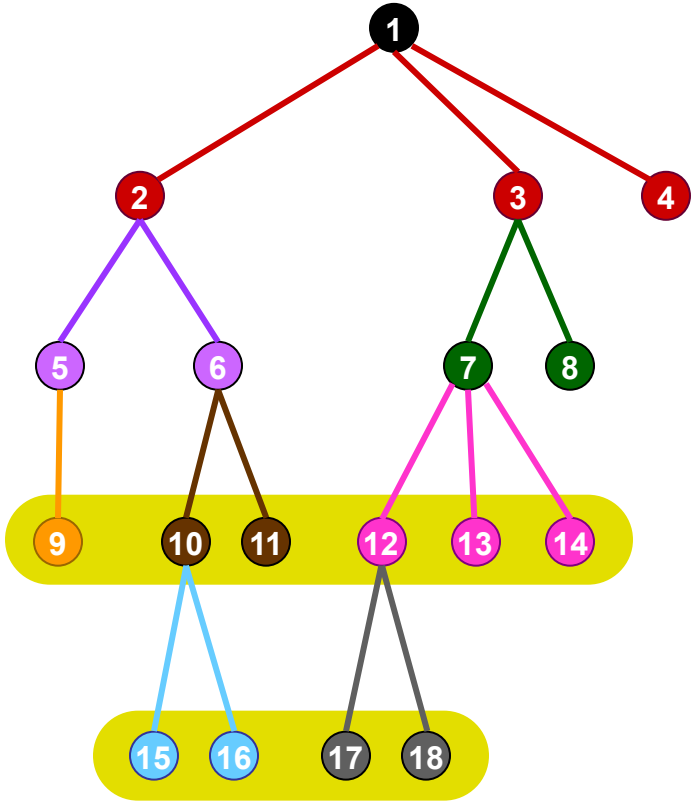
# How to make rainbows

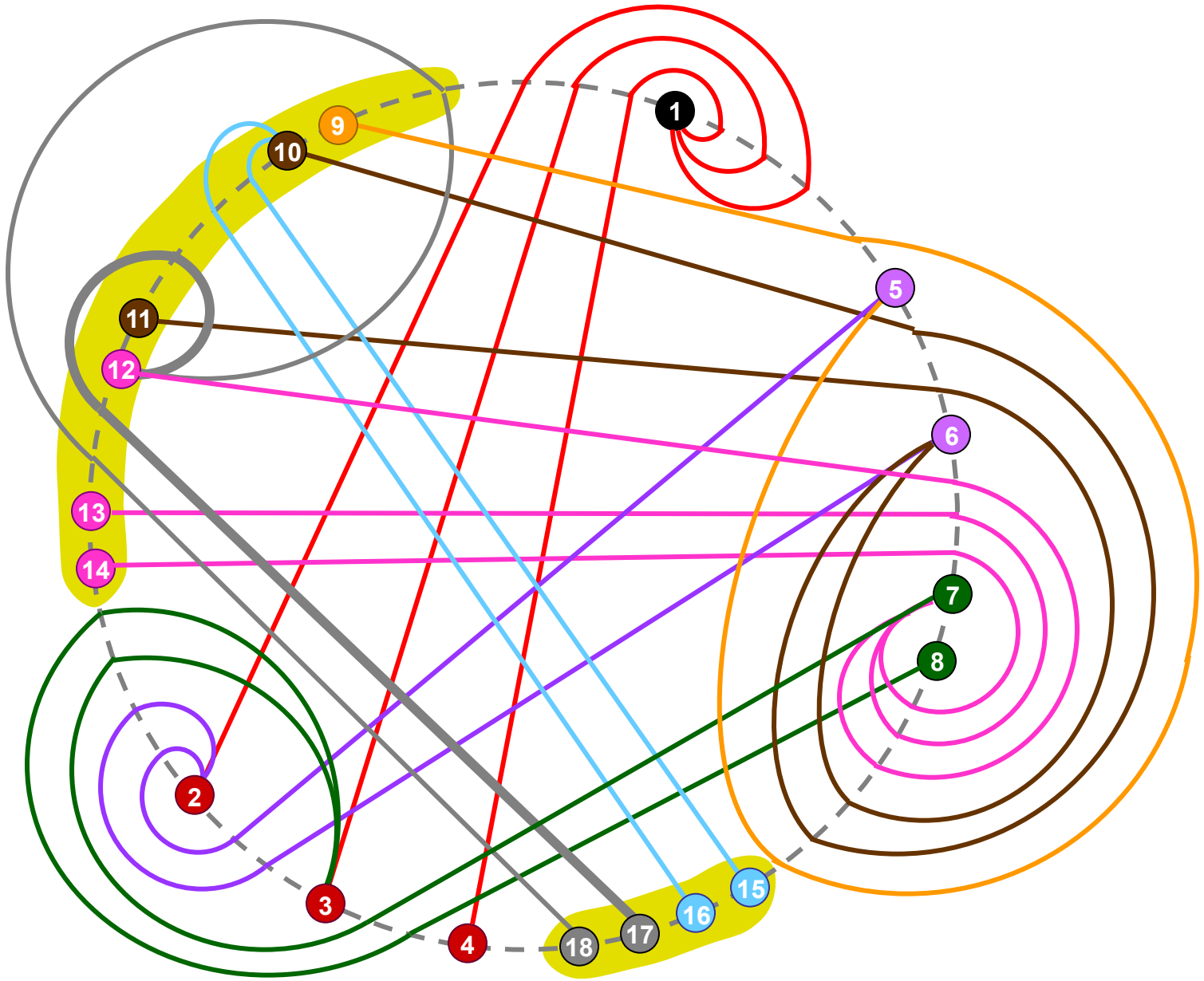


# How to make rainbows

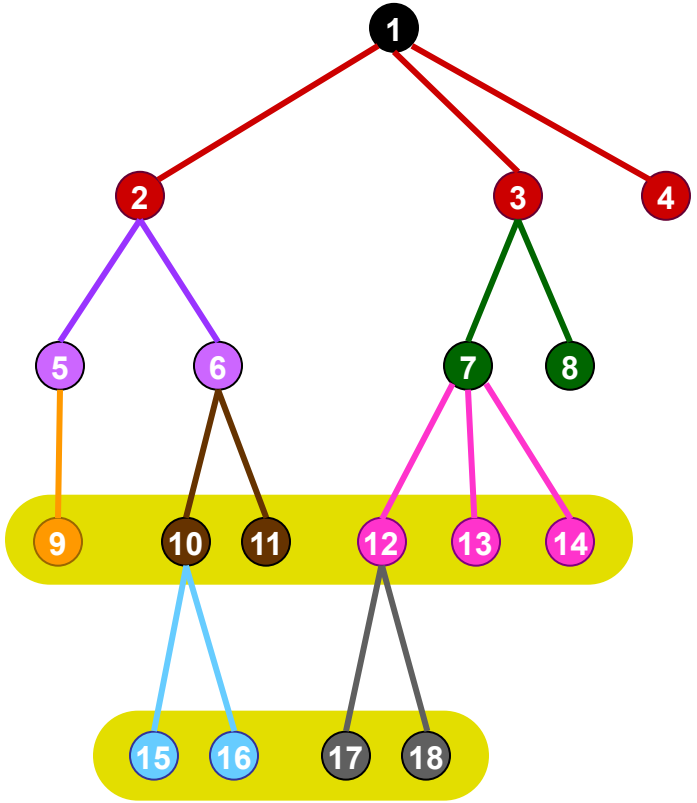
**WARNING**: removed 3 crossings instead of 1 !!!

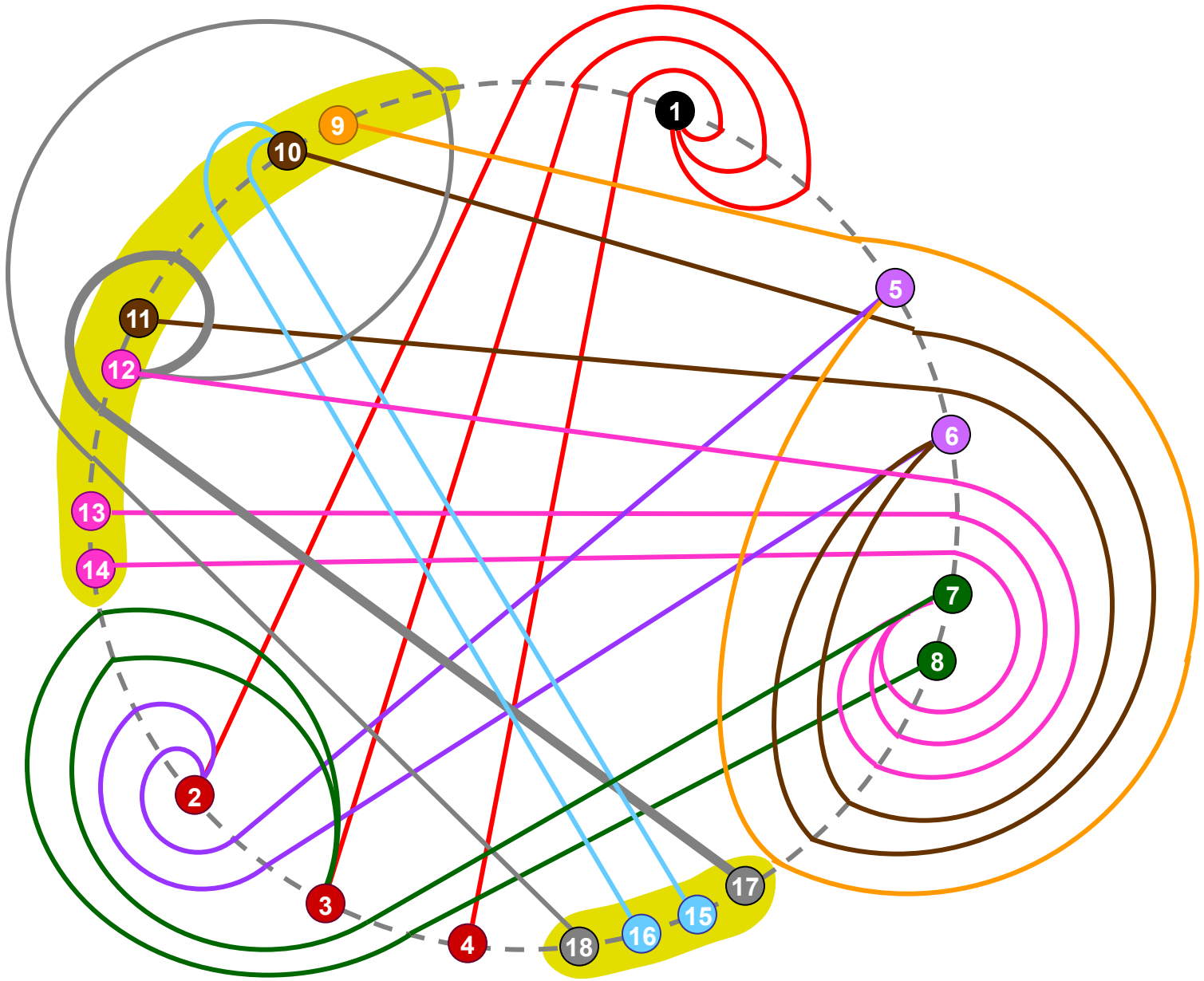


# How to make rainbows

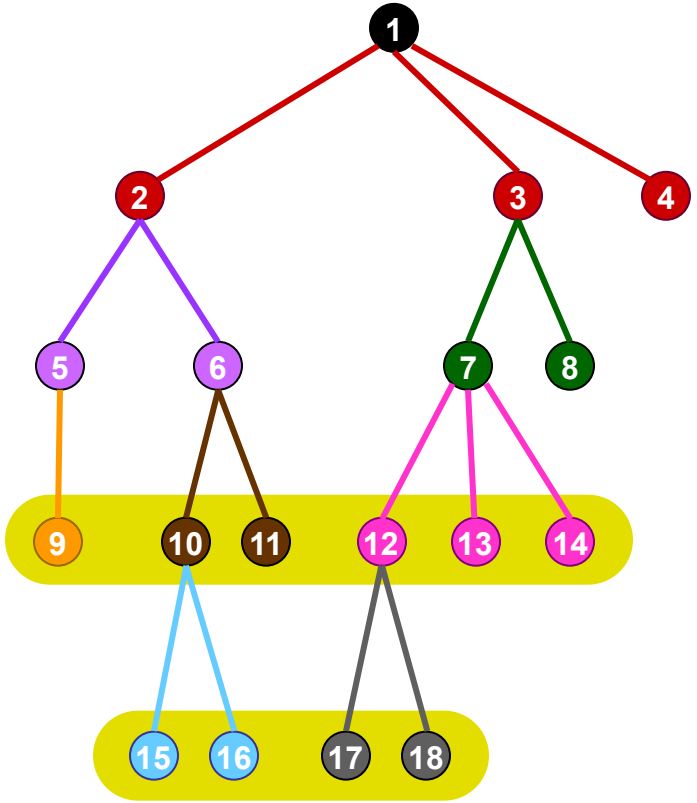
**FIX**: reinsert 2 crossings with the lower level

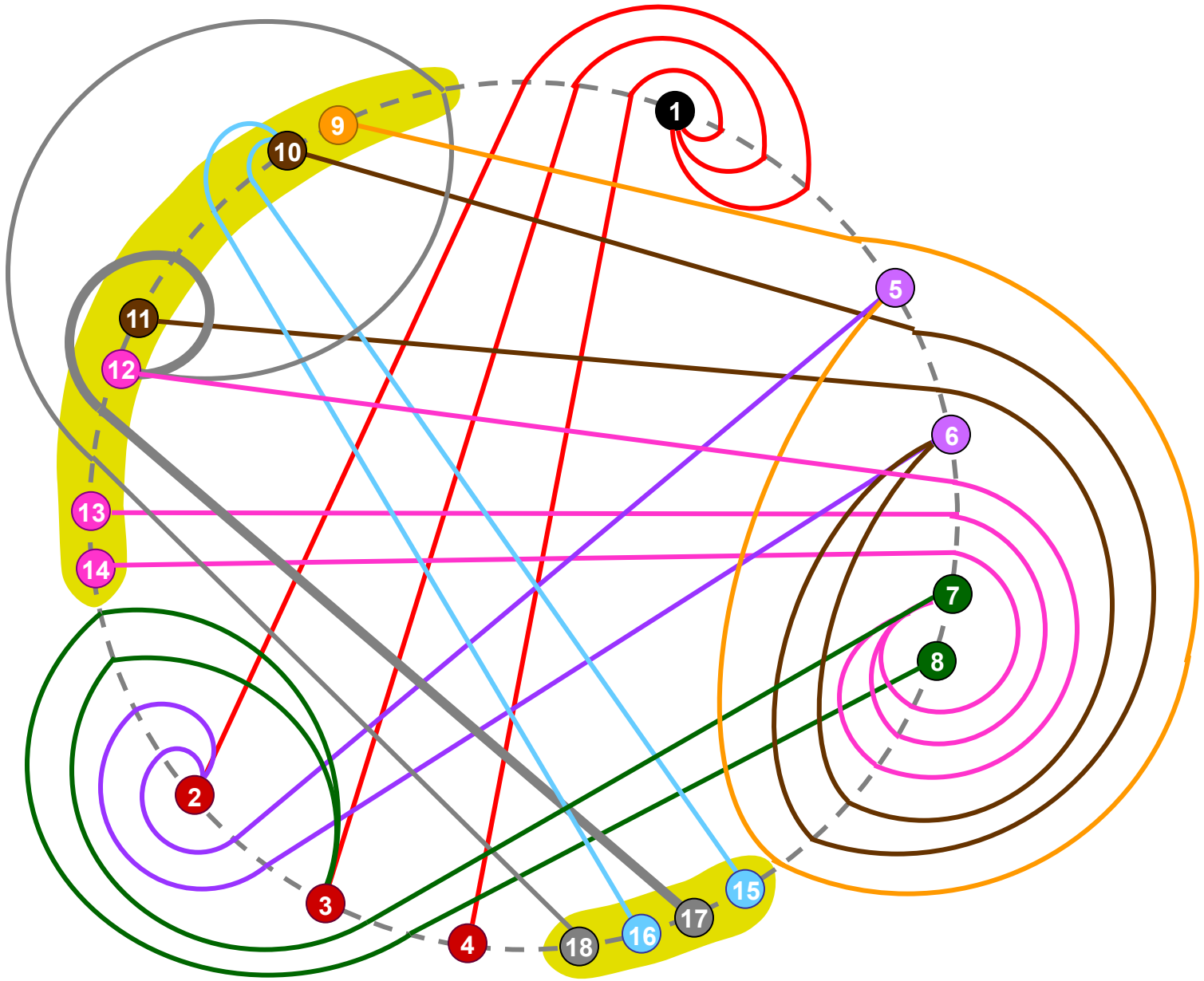


# How to make rainbows

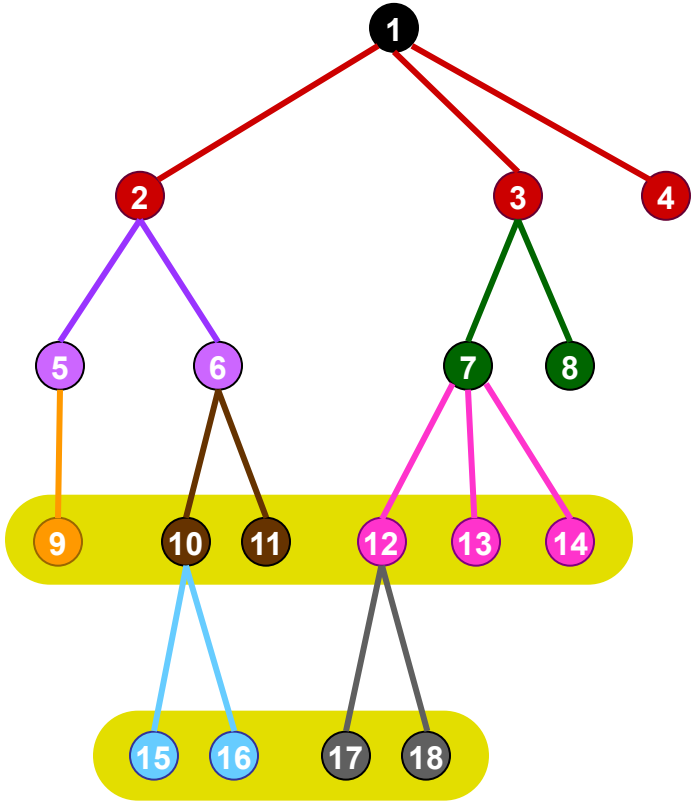
…remove one of them….

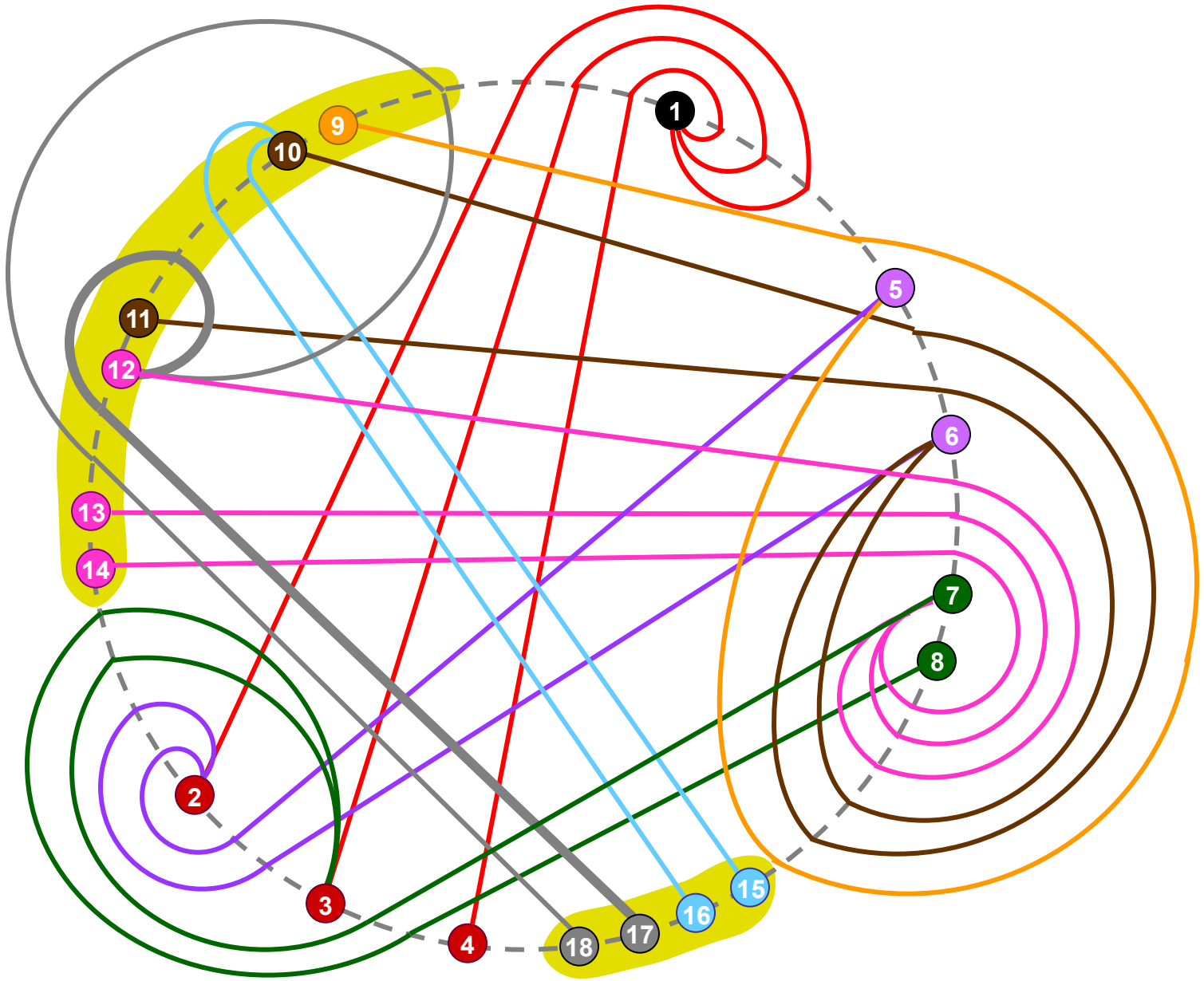


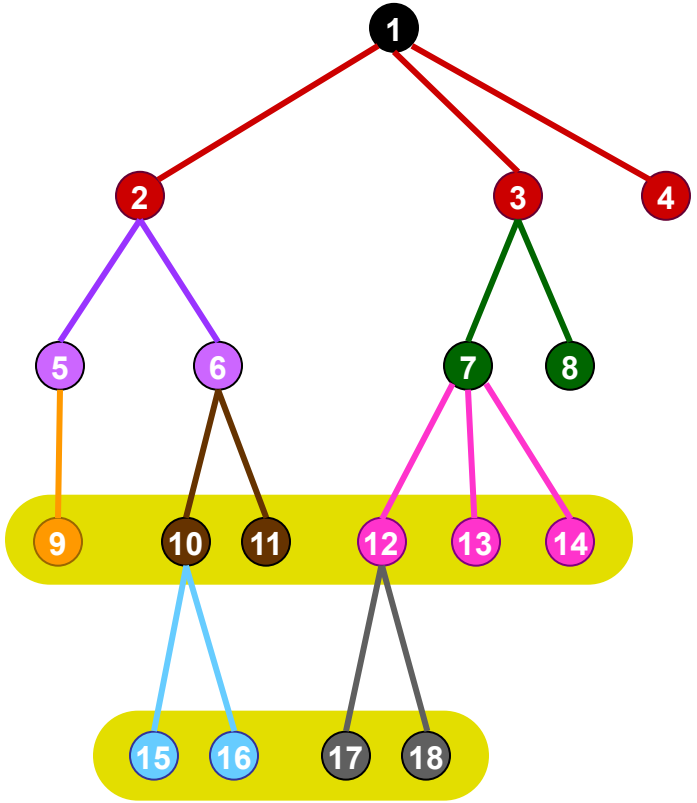
# How to make rainbows

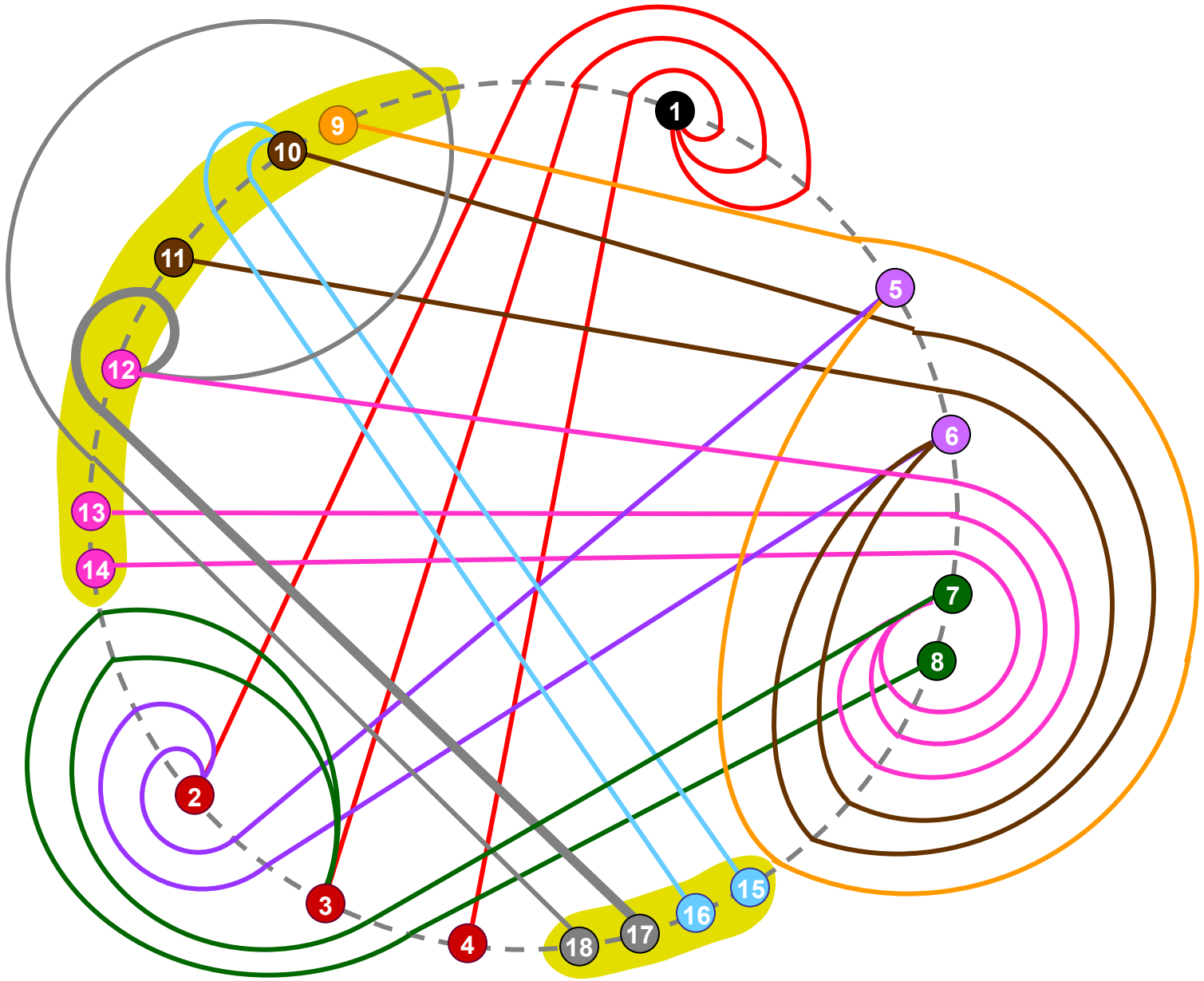
…remove the other one….



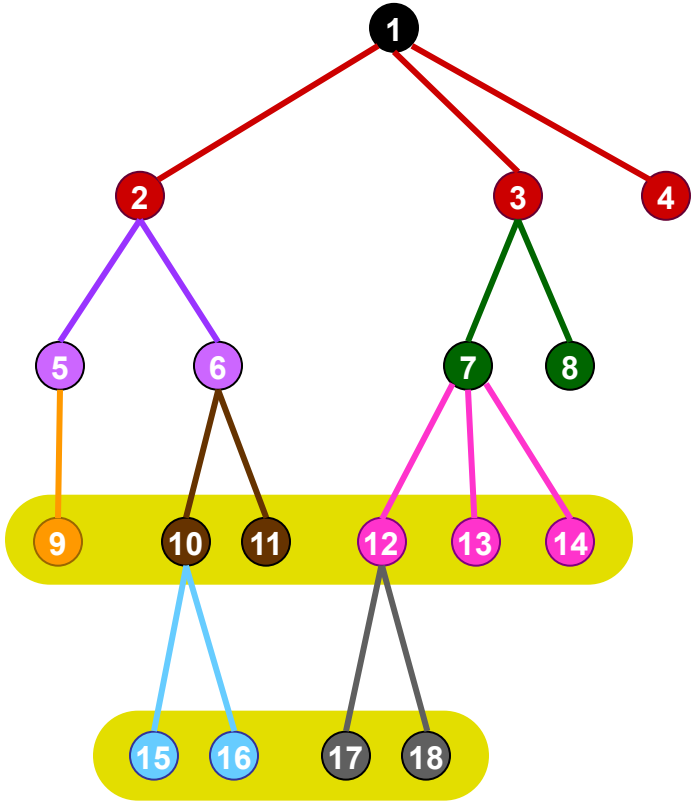



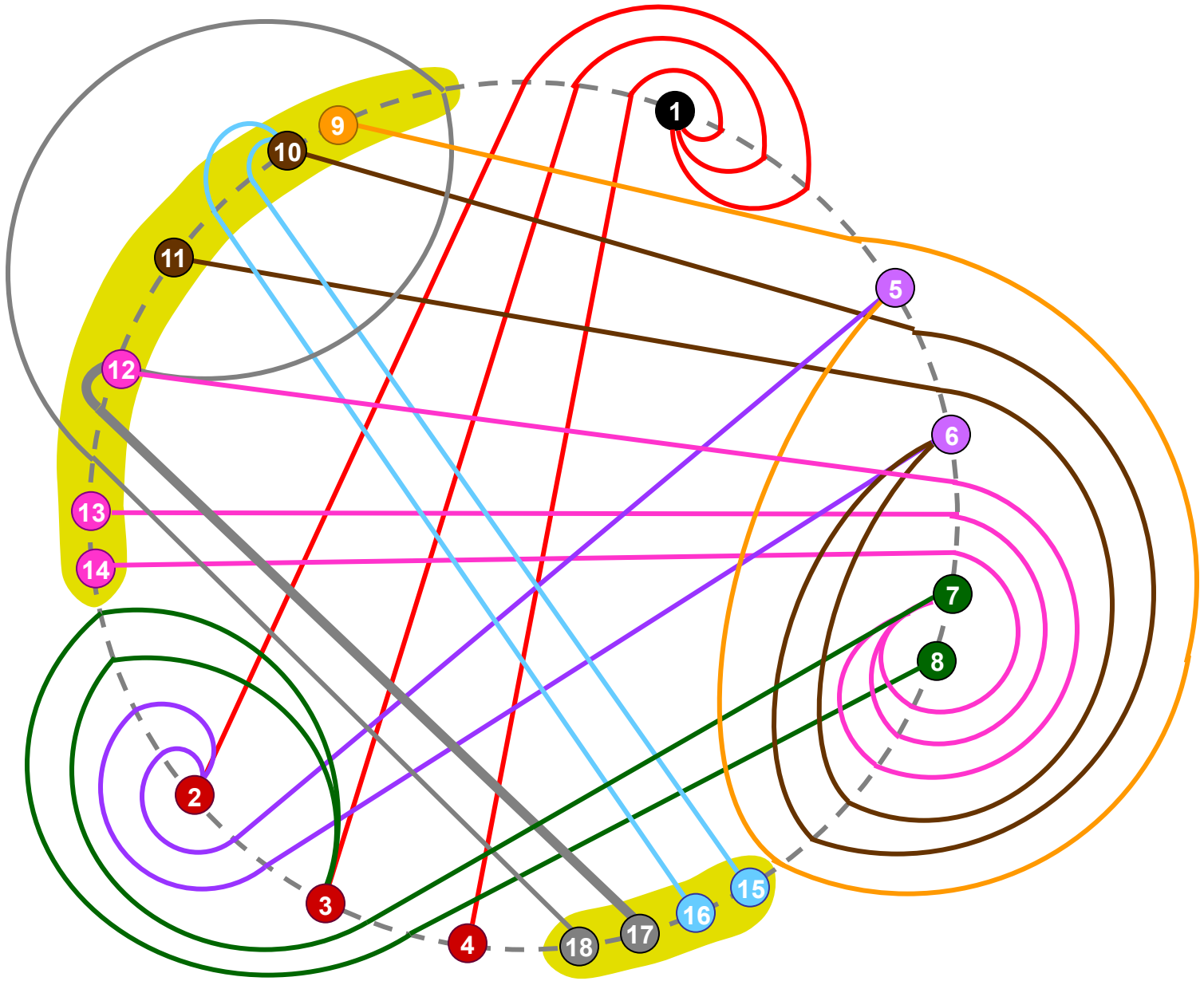
# How to make rainbows



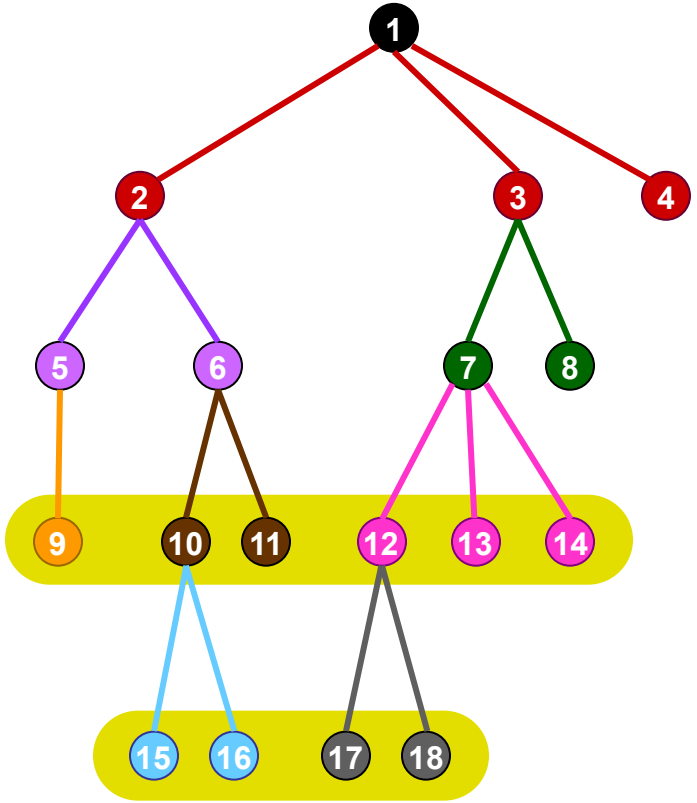



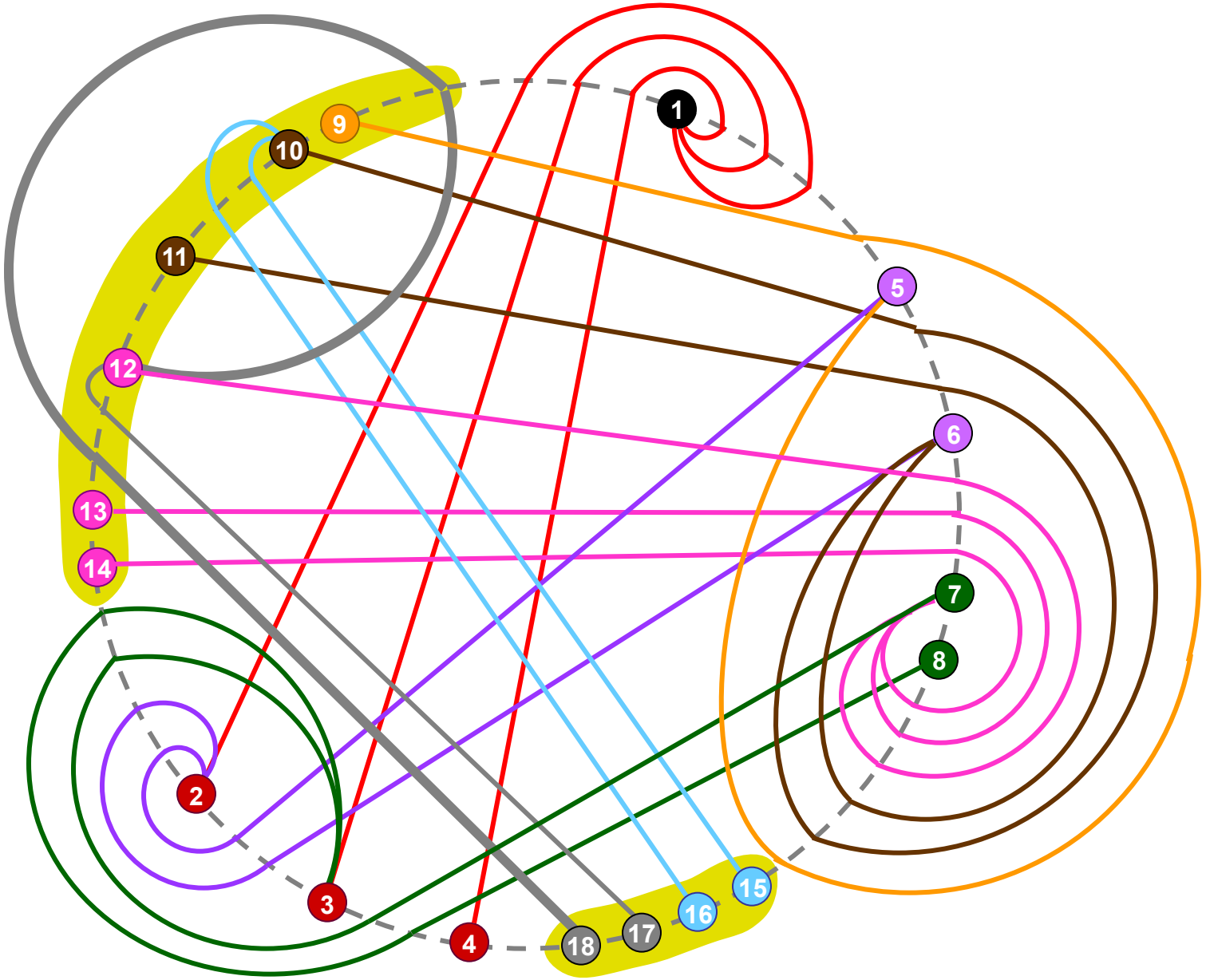
# How to make rainbows



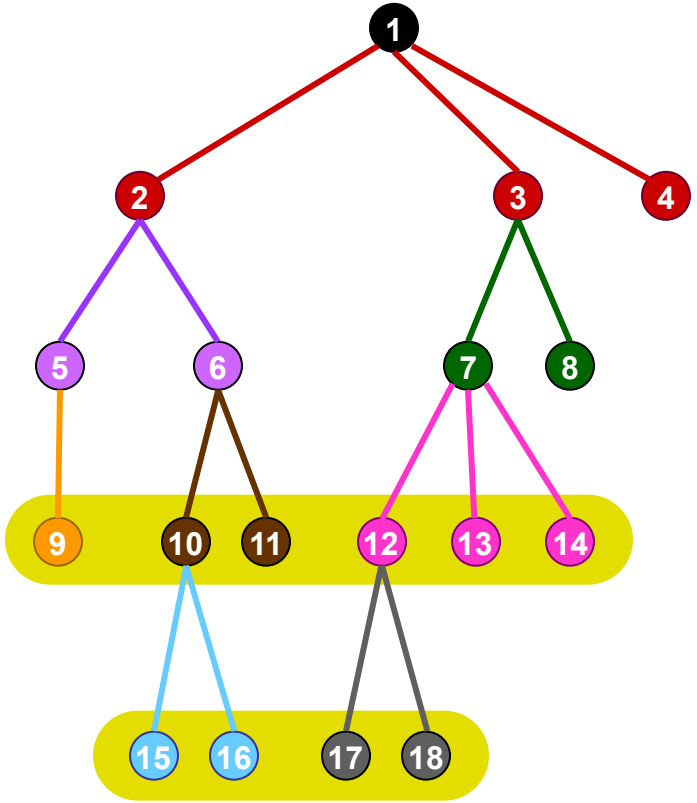



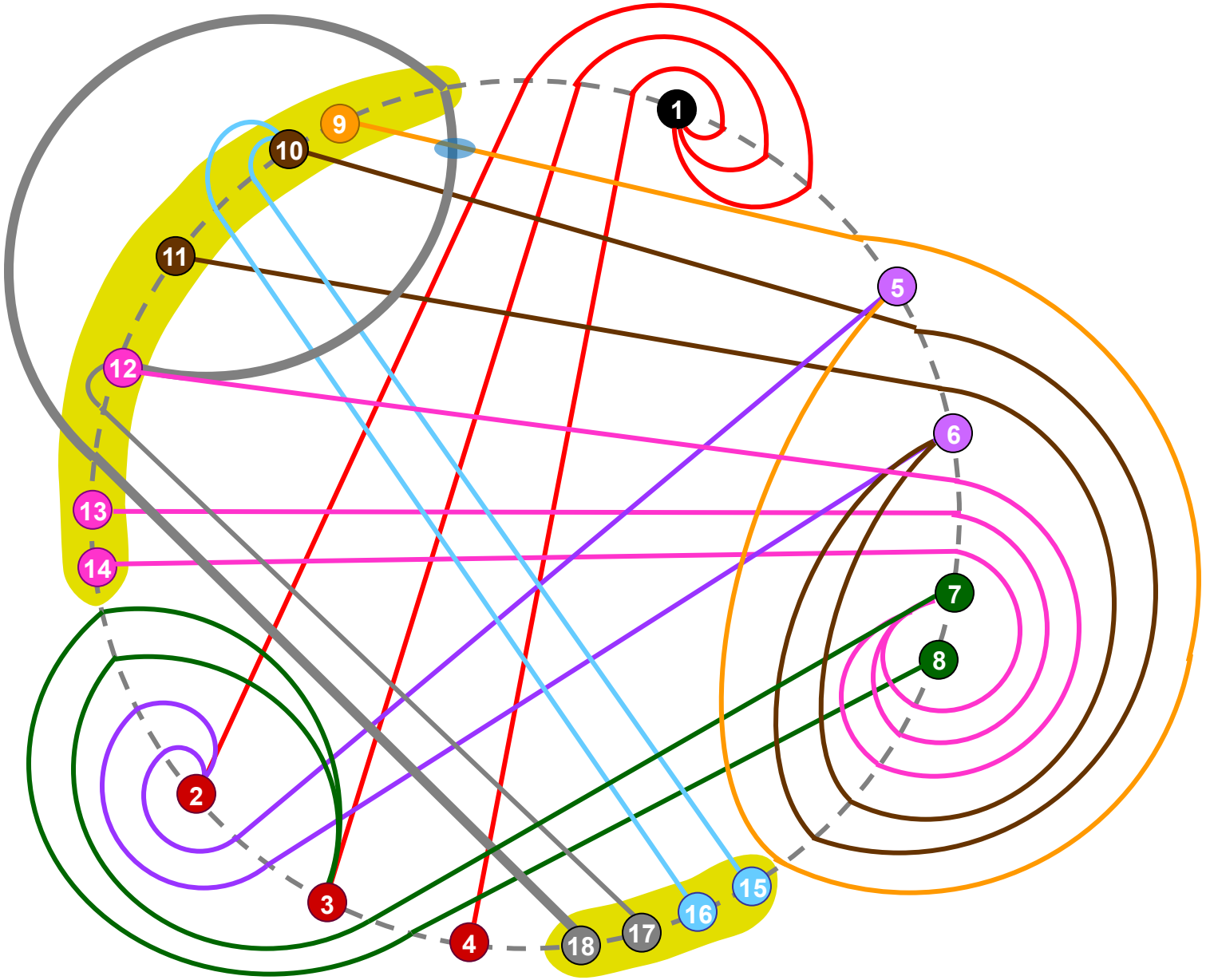
# How to make rainbows



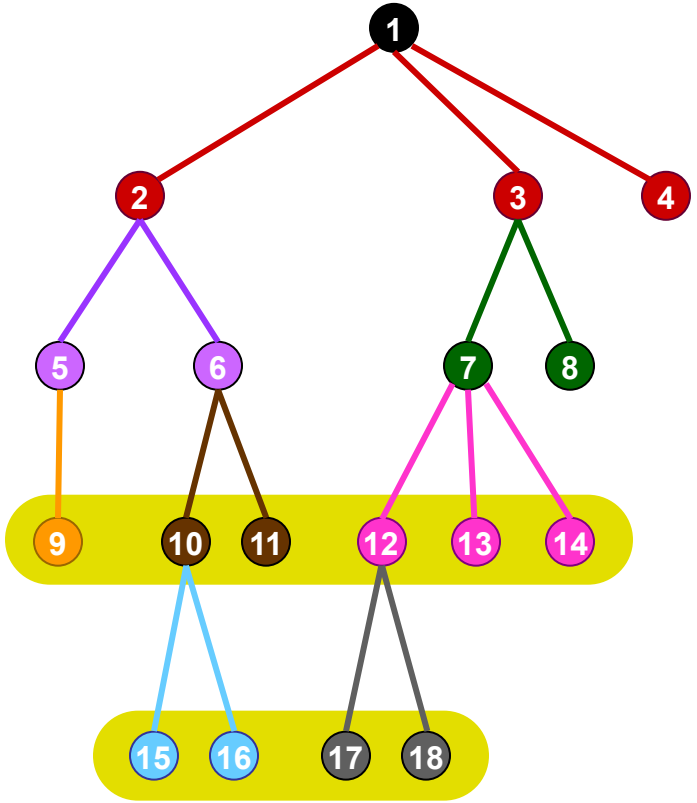



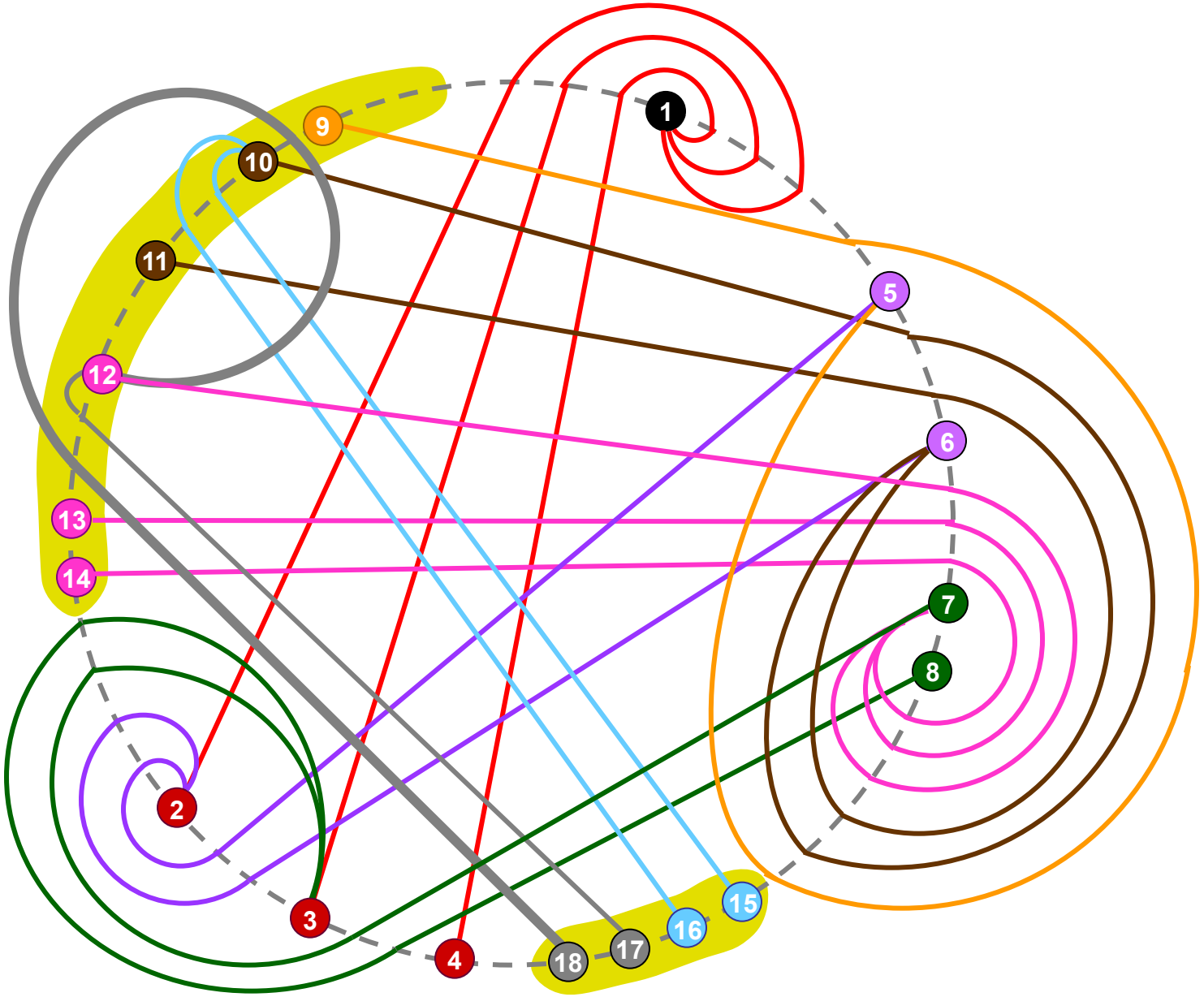
# How to make rainbows



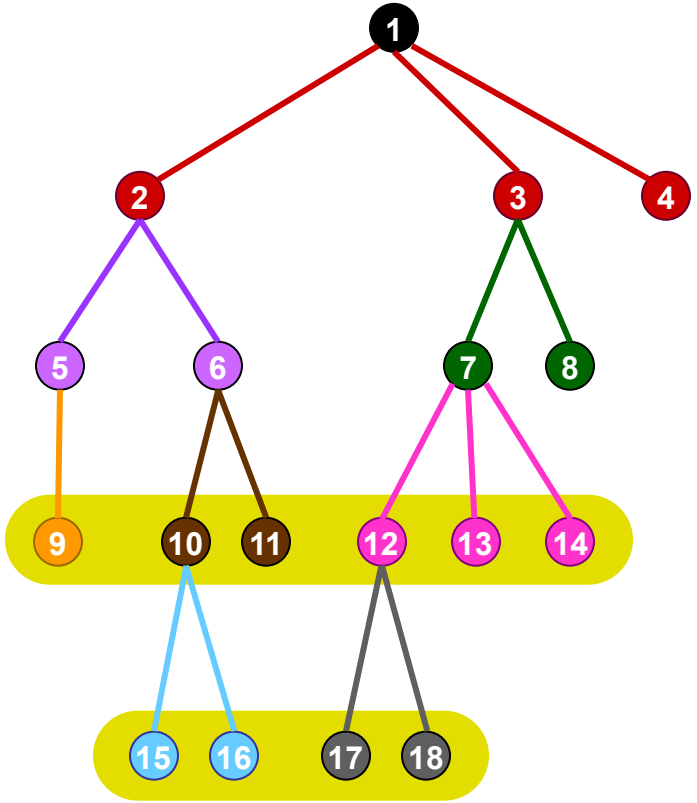



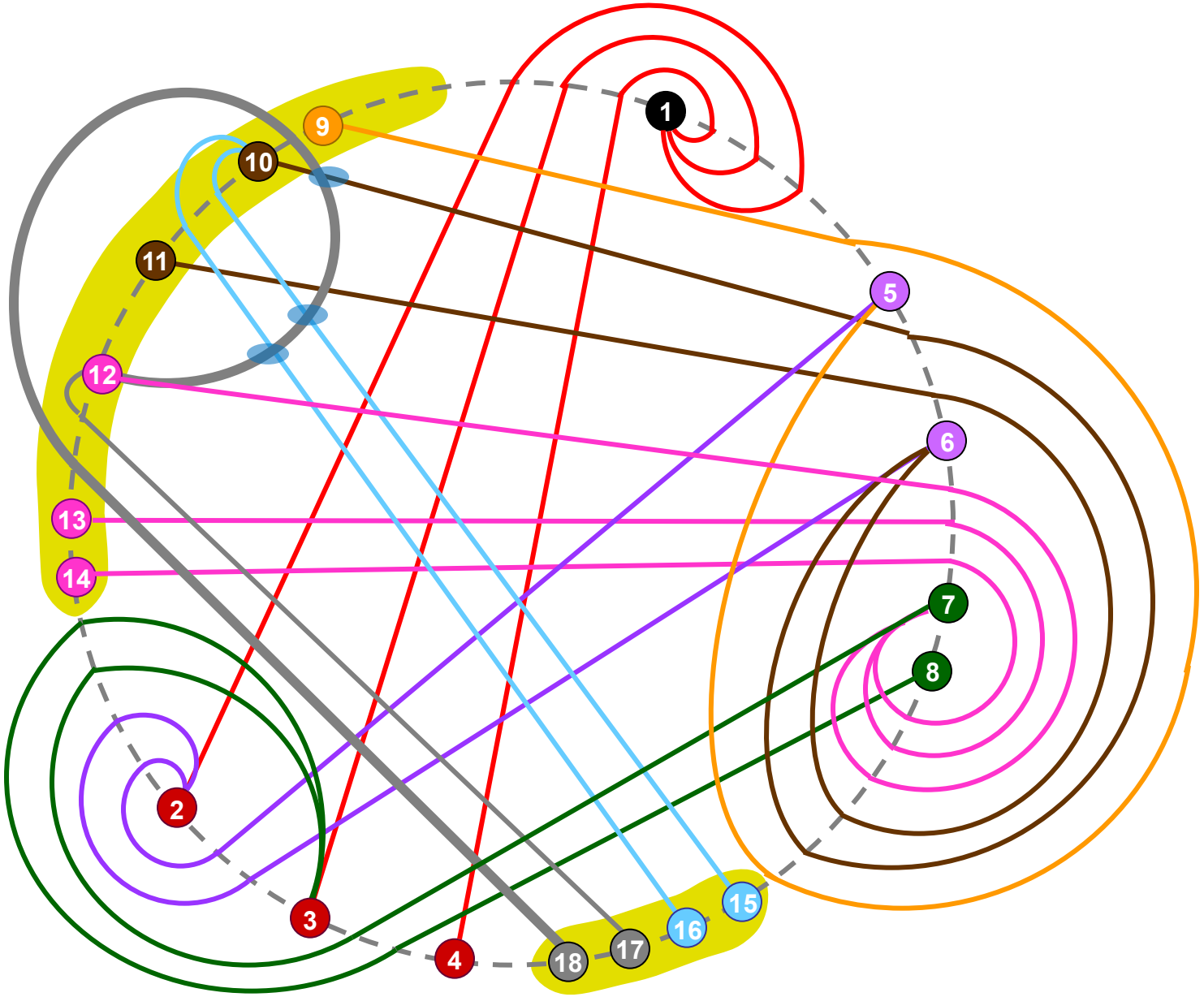
# How to make rainbows



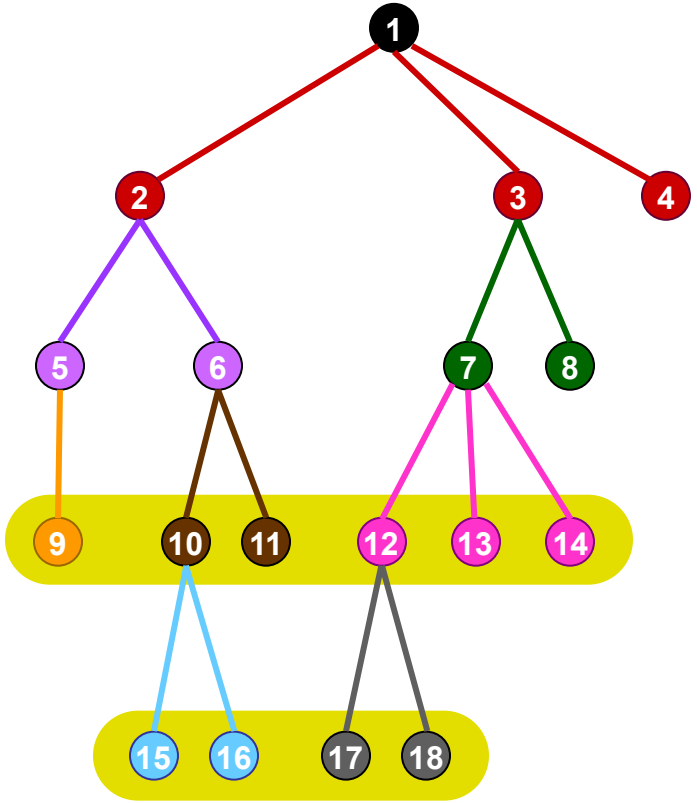
# How to make rainbows

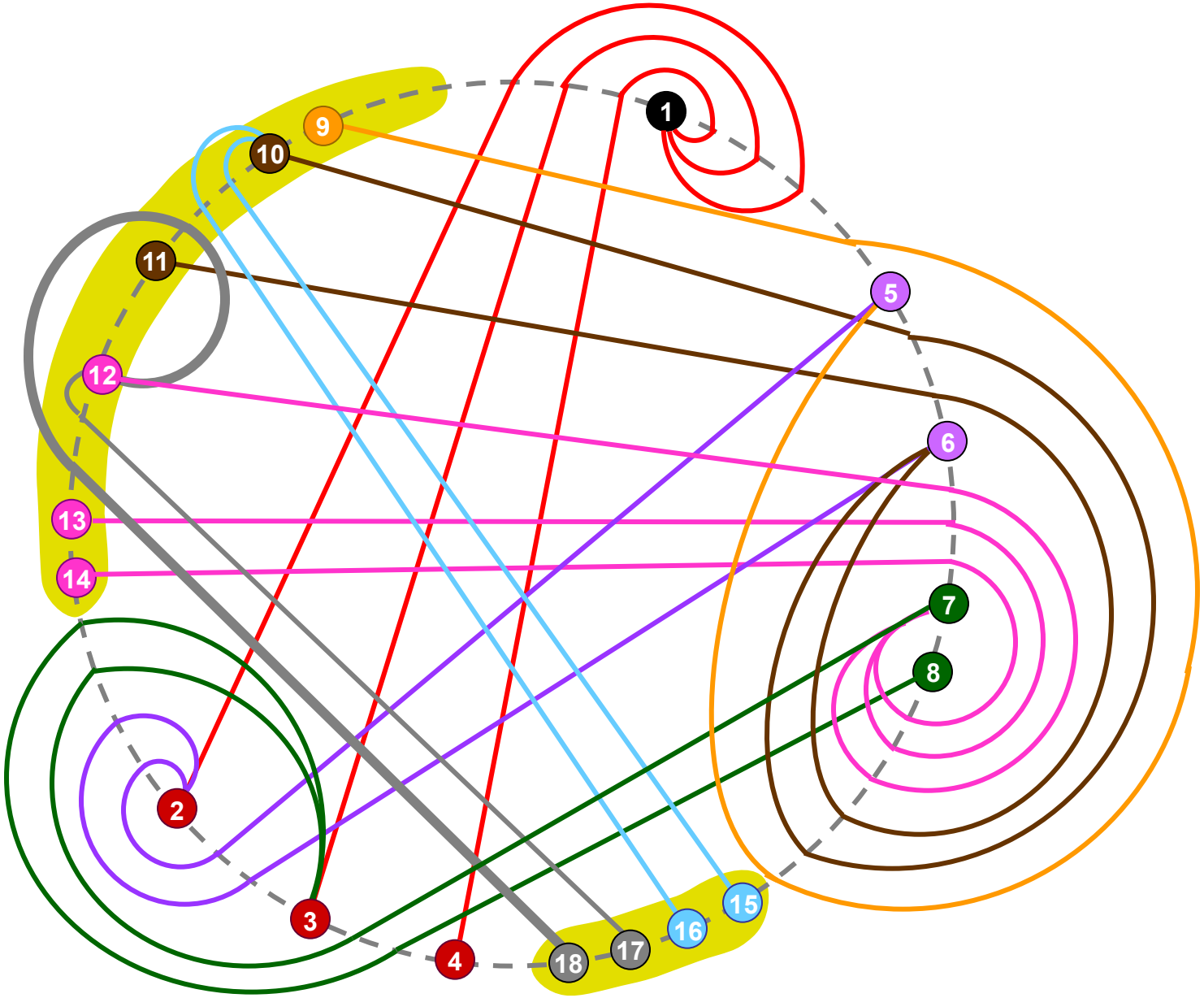


# How to make rainbows

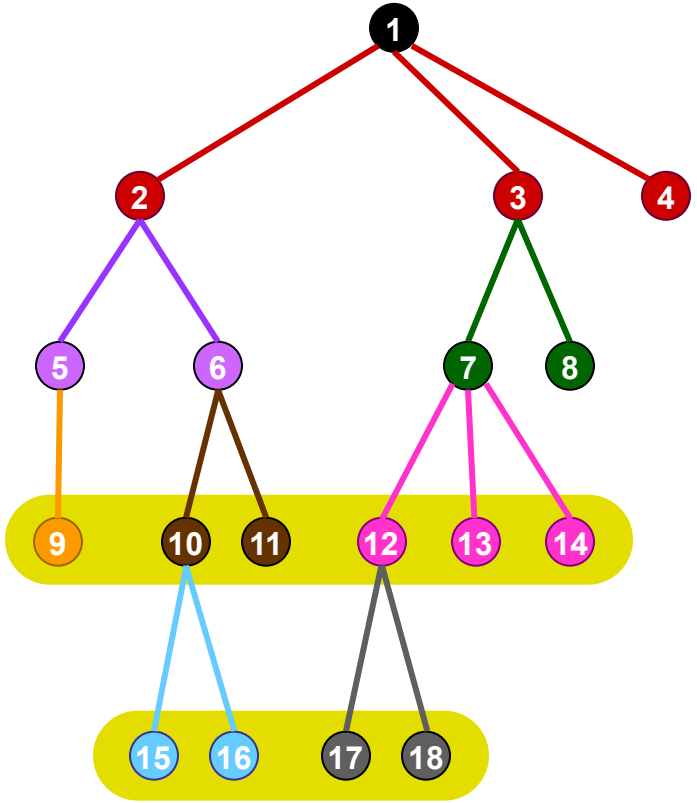
**WARNING**: removed 3 crossings instead of 1 !!!

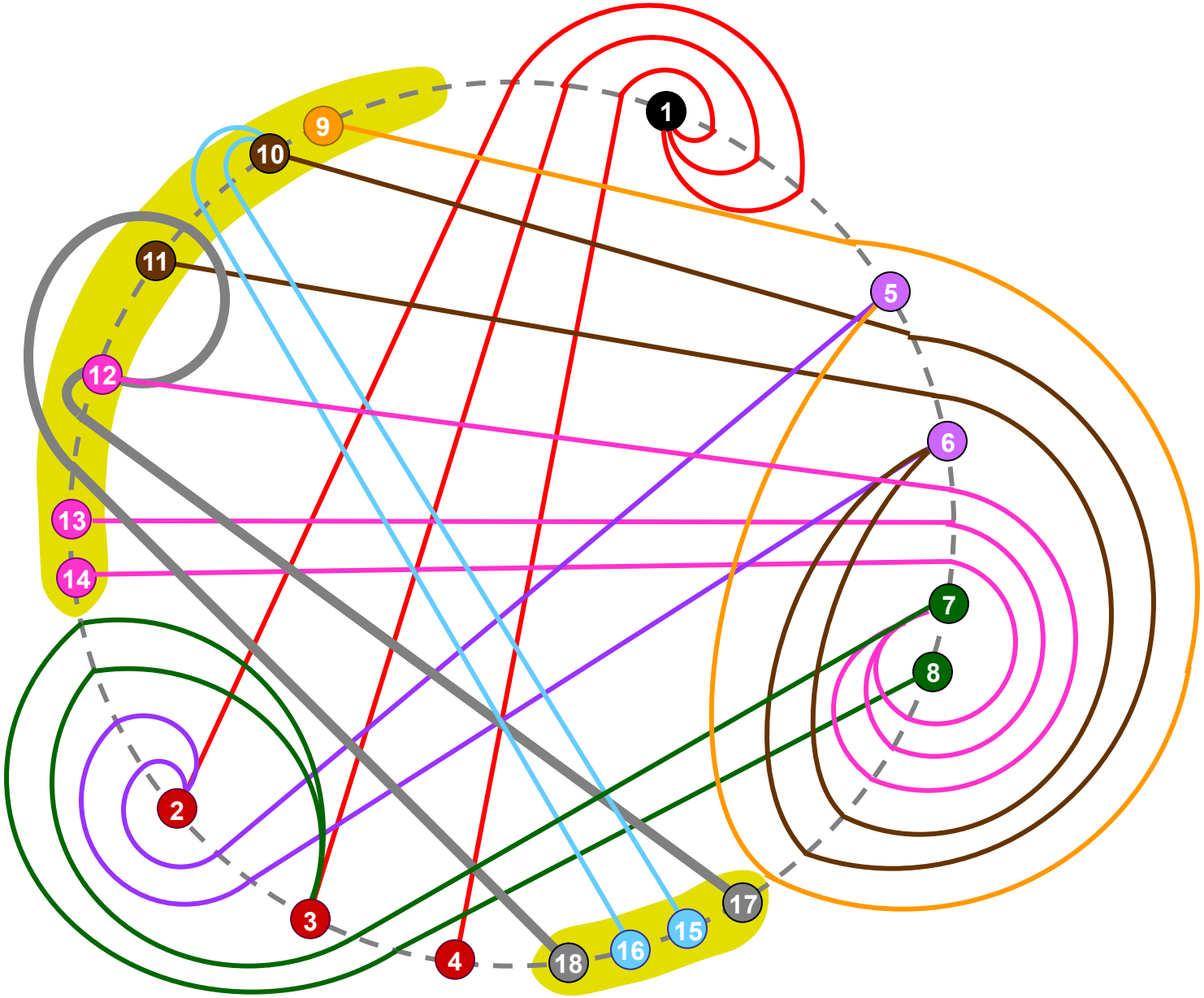


# How to make rainbows

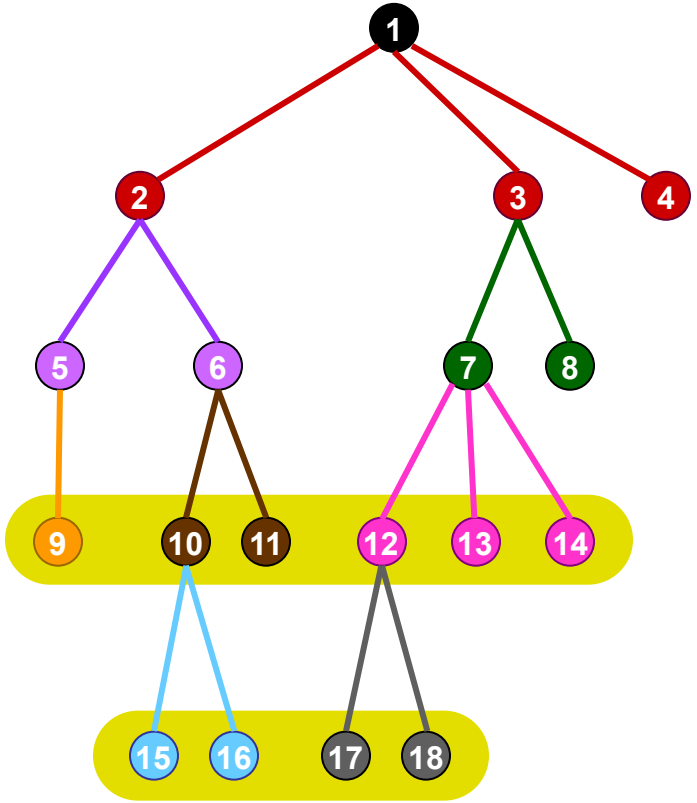
**FIX**: reinsert 2 crossings with the lower level

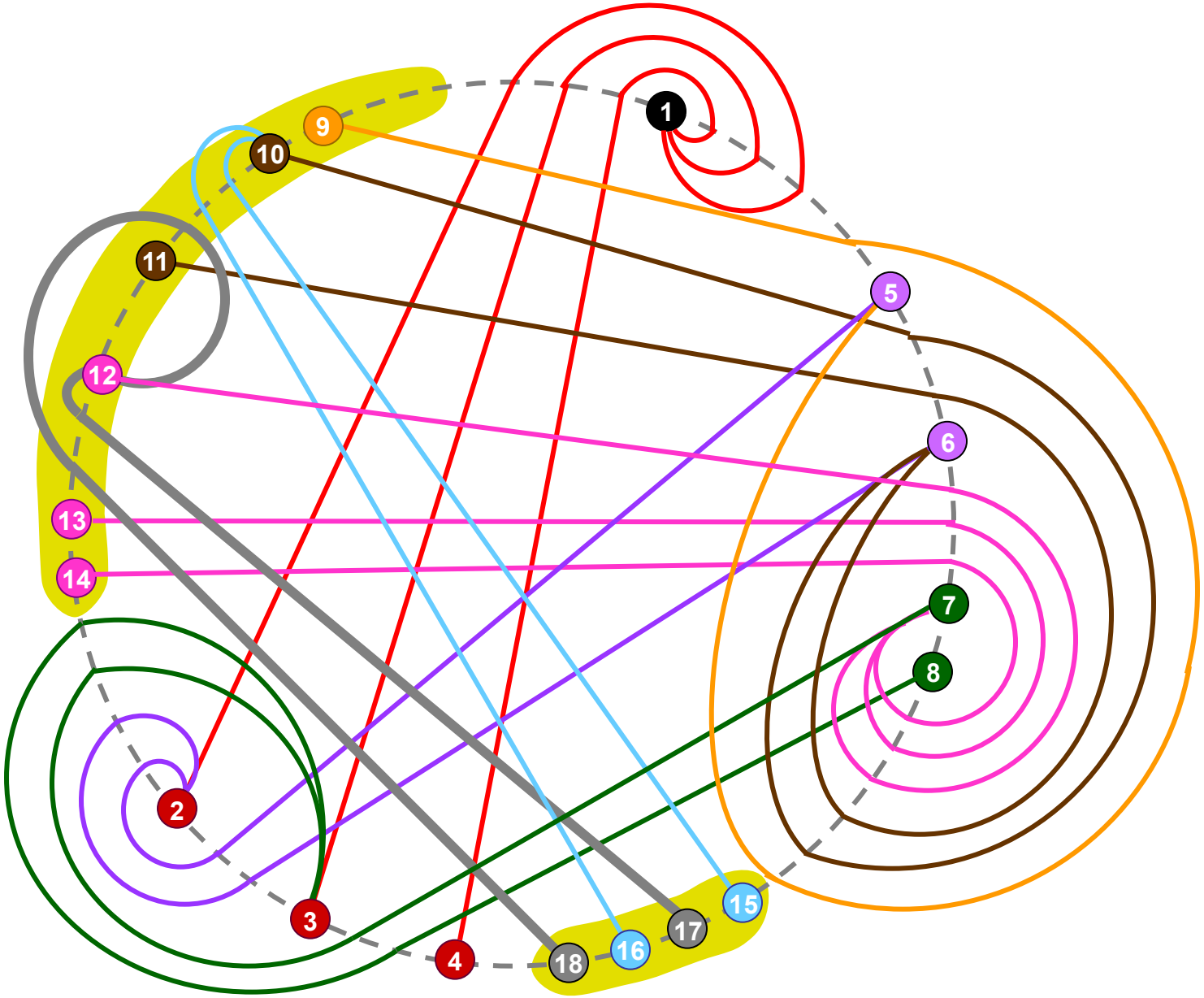


# How to make rainbows

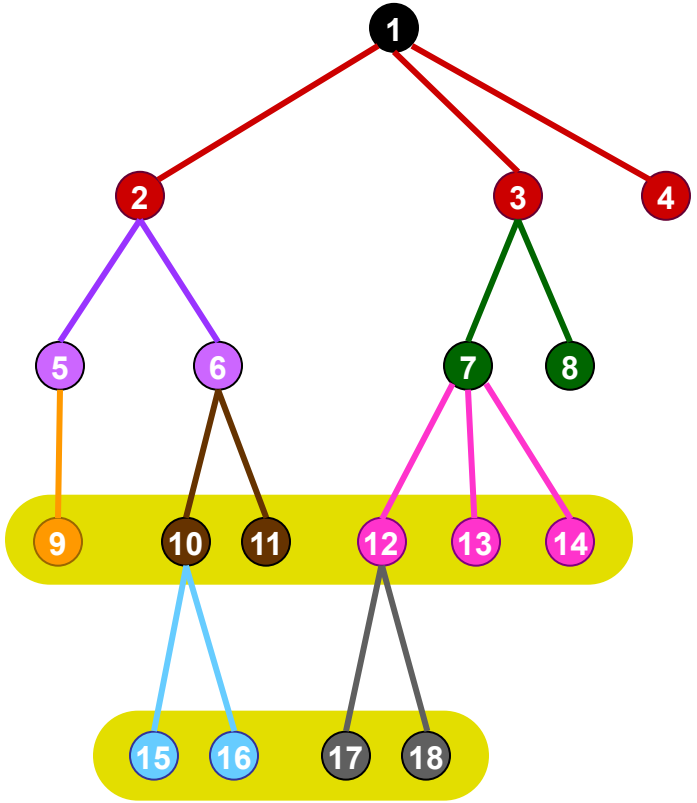
…remove one of them…

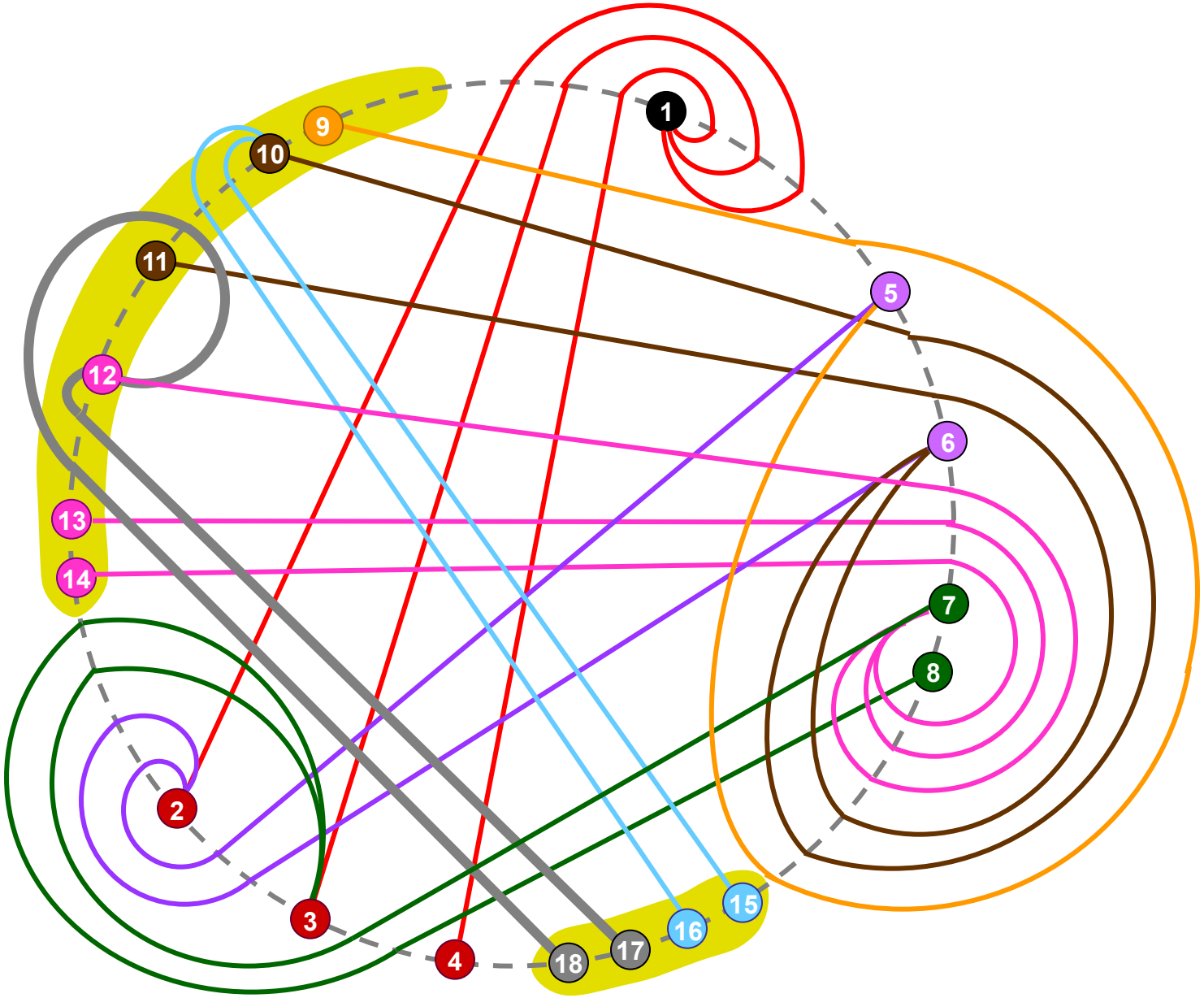


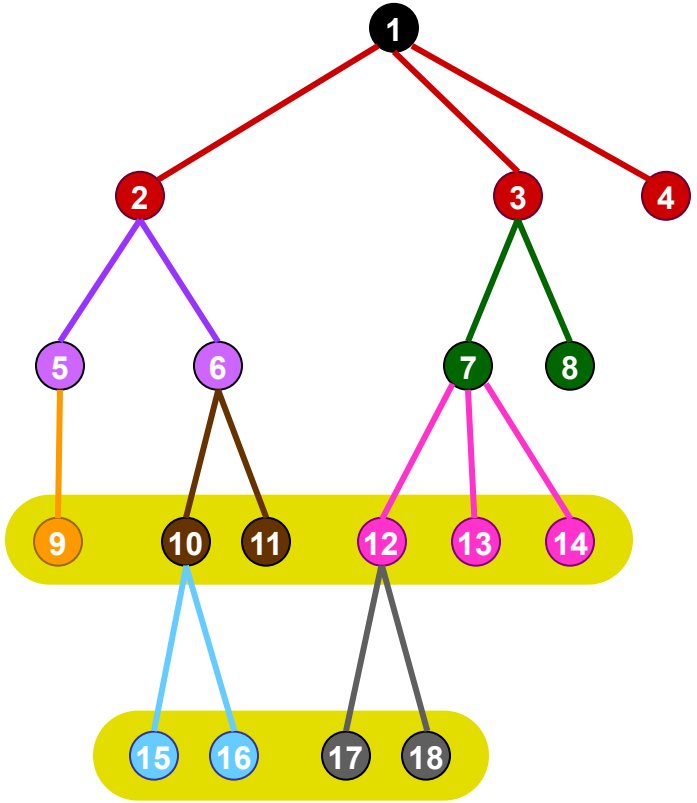
# How to make rainbows

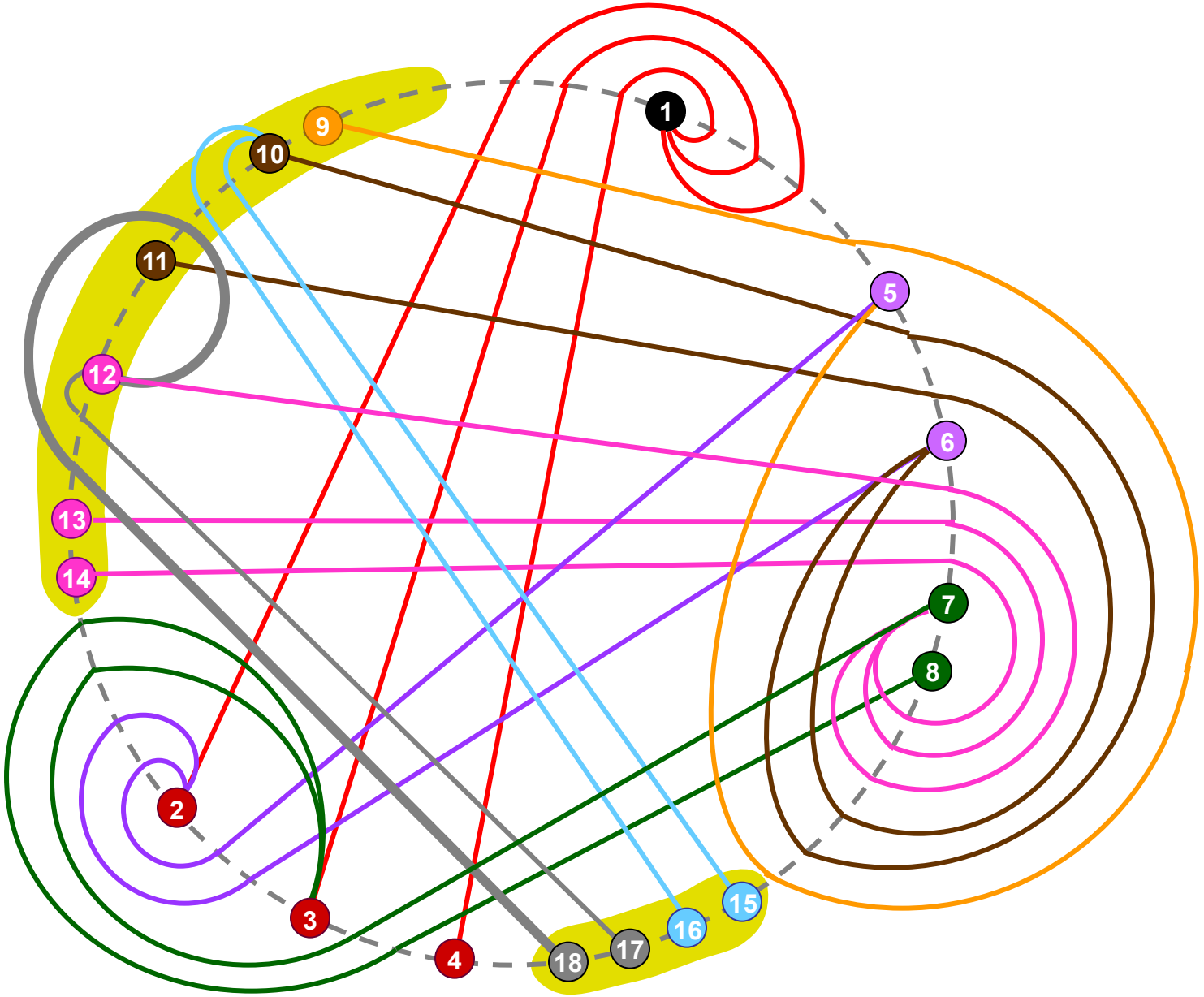
…remove the other one…



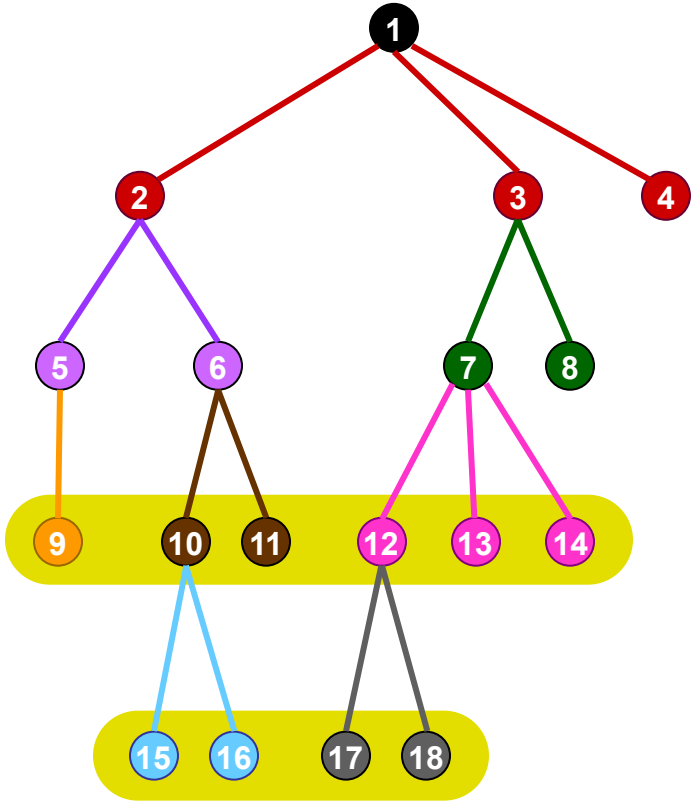



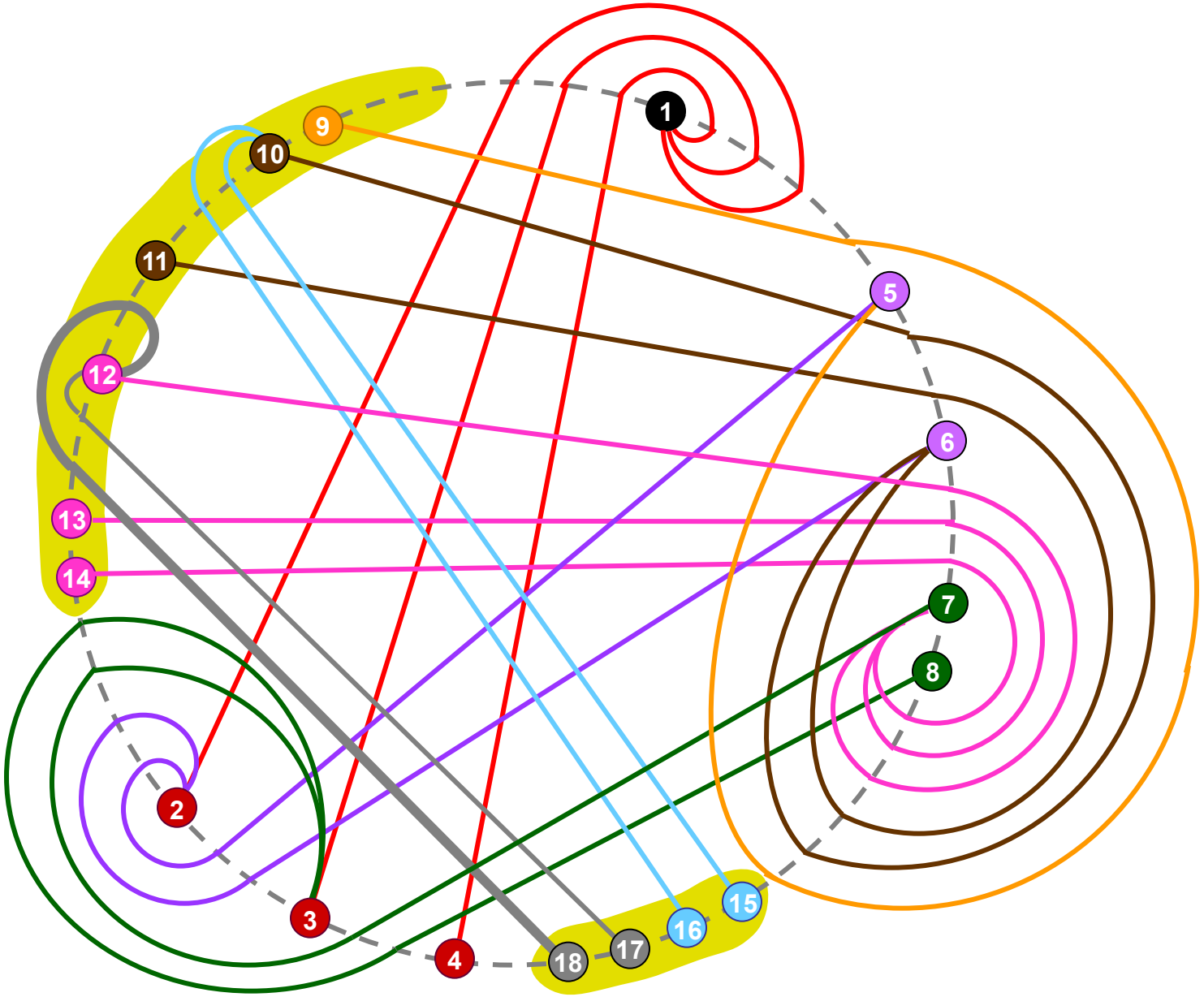
# How to make rainbows



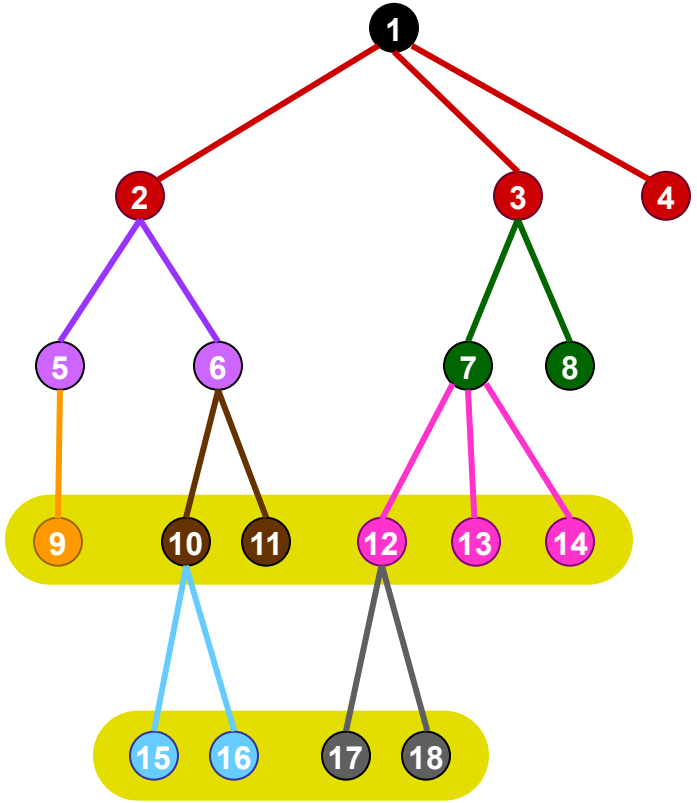



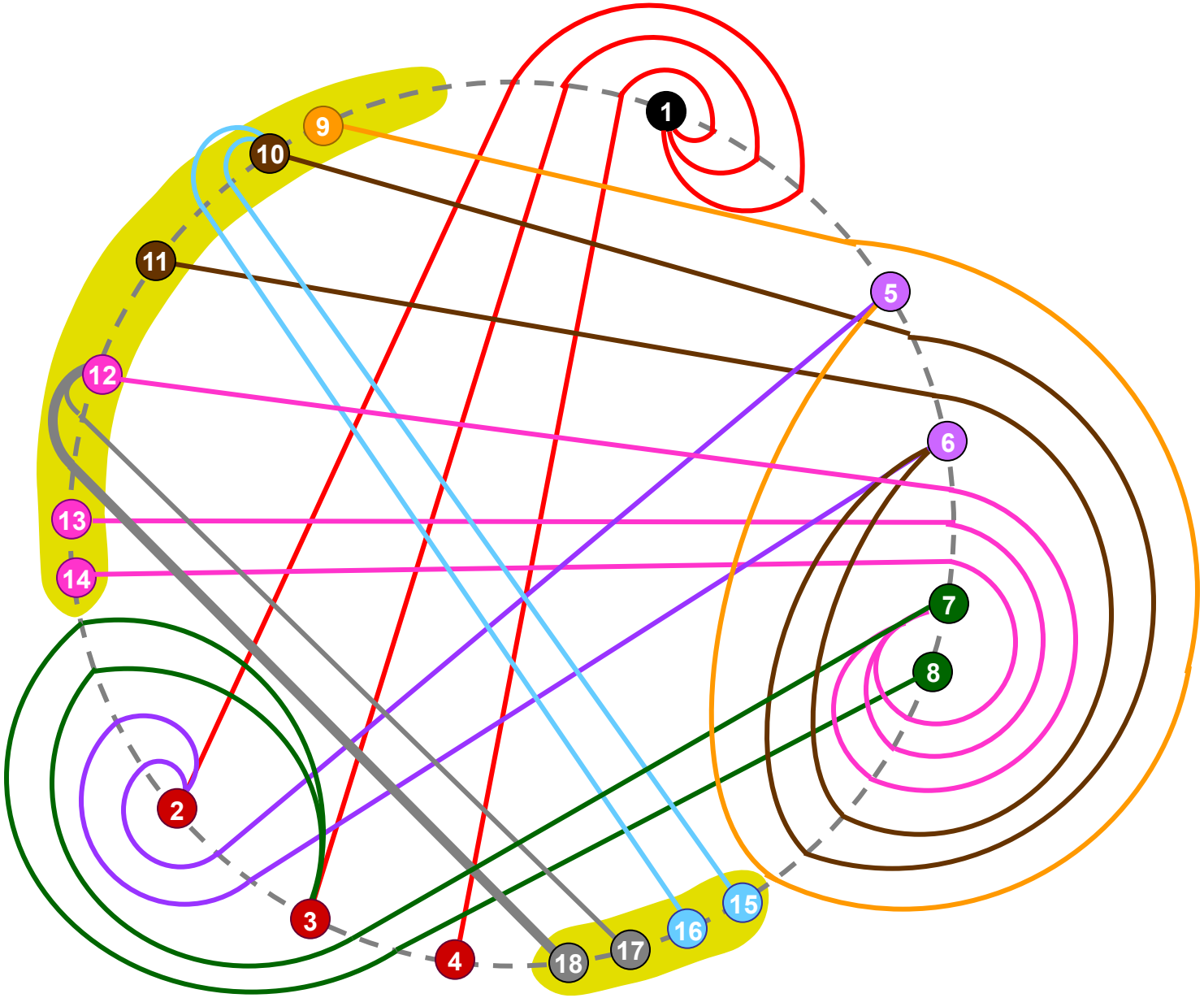
# How to make rainbows



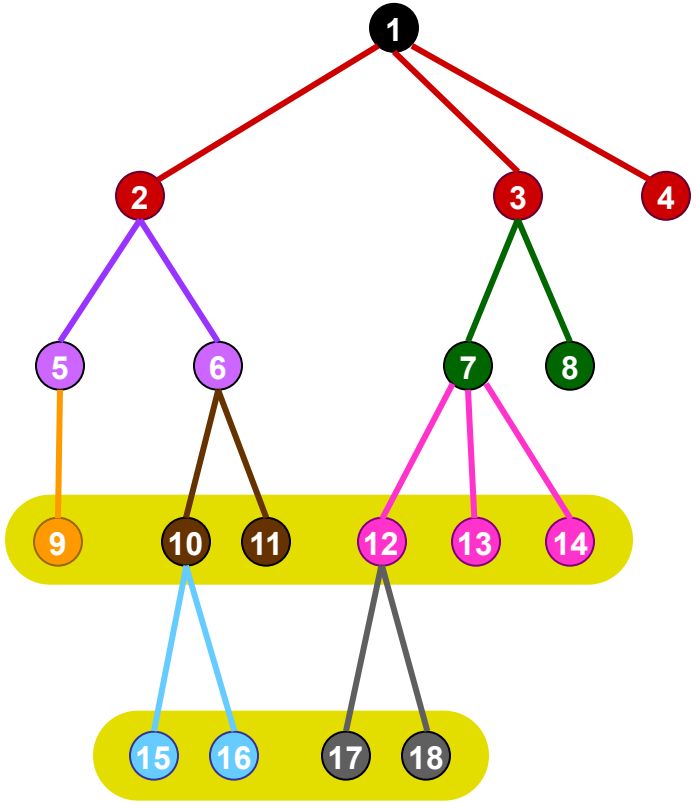



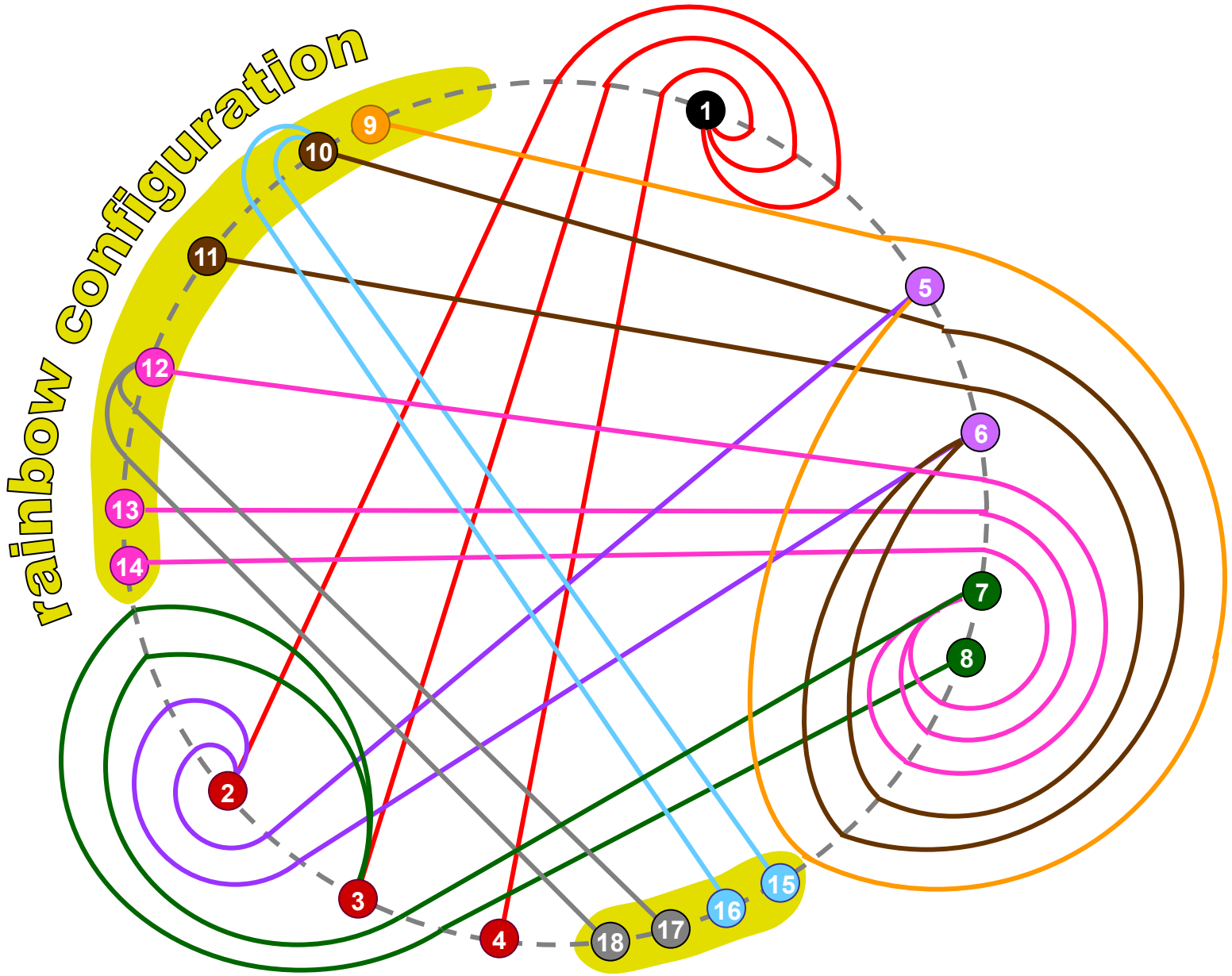
# How to make rainbows



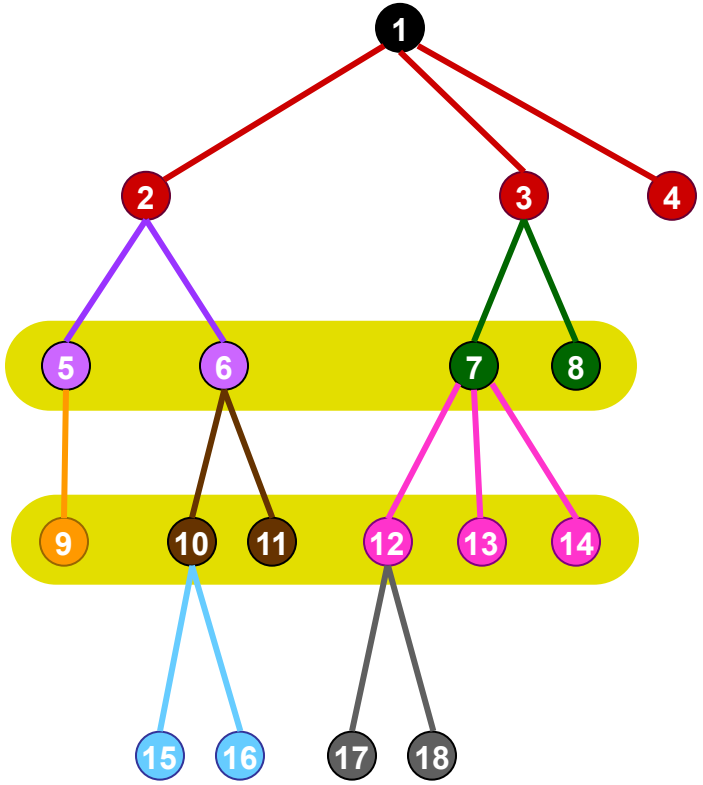



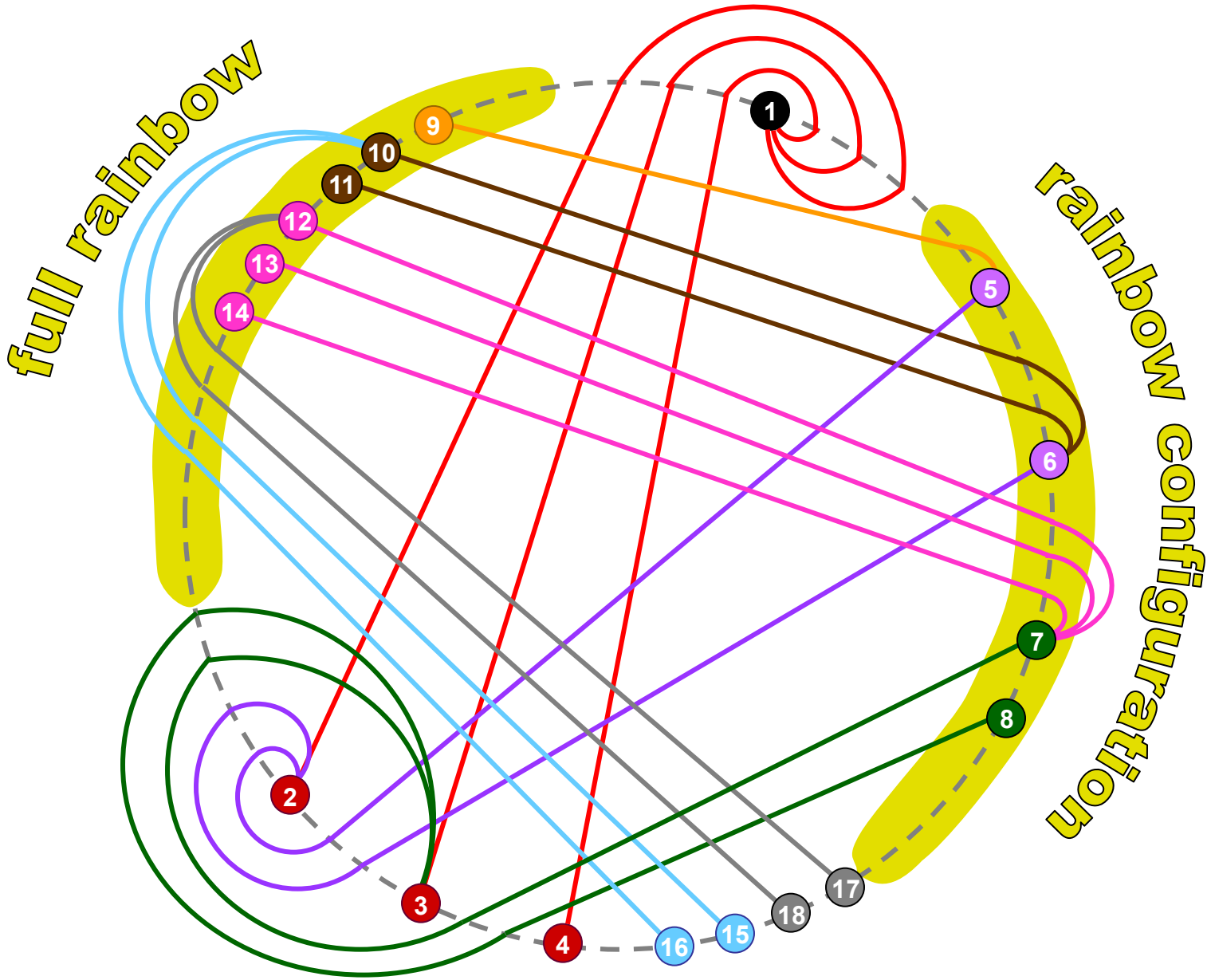
# How to make rainbows



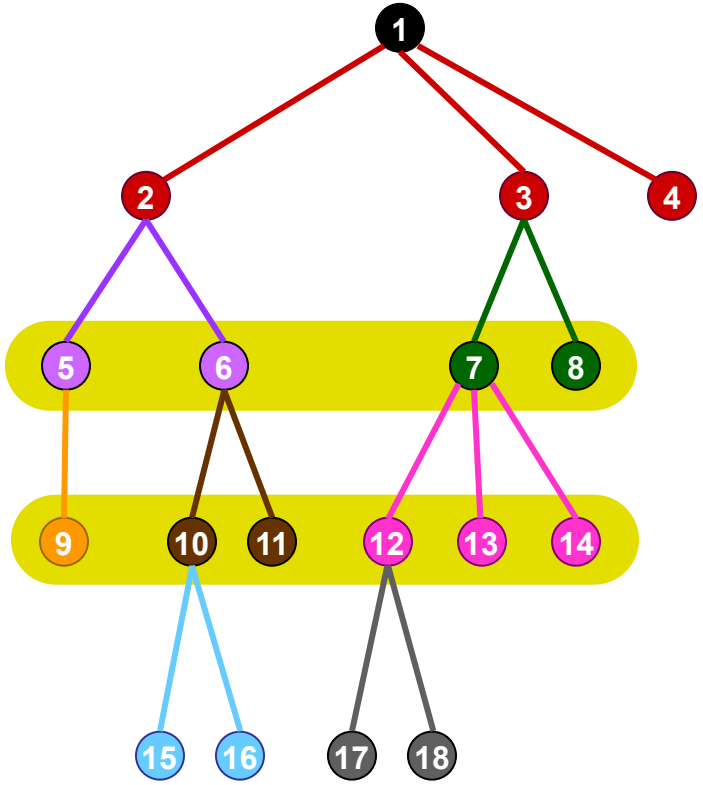



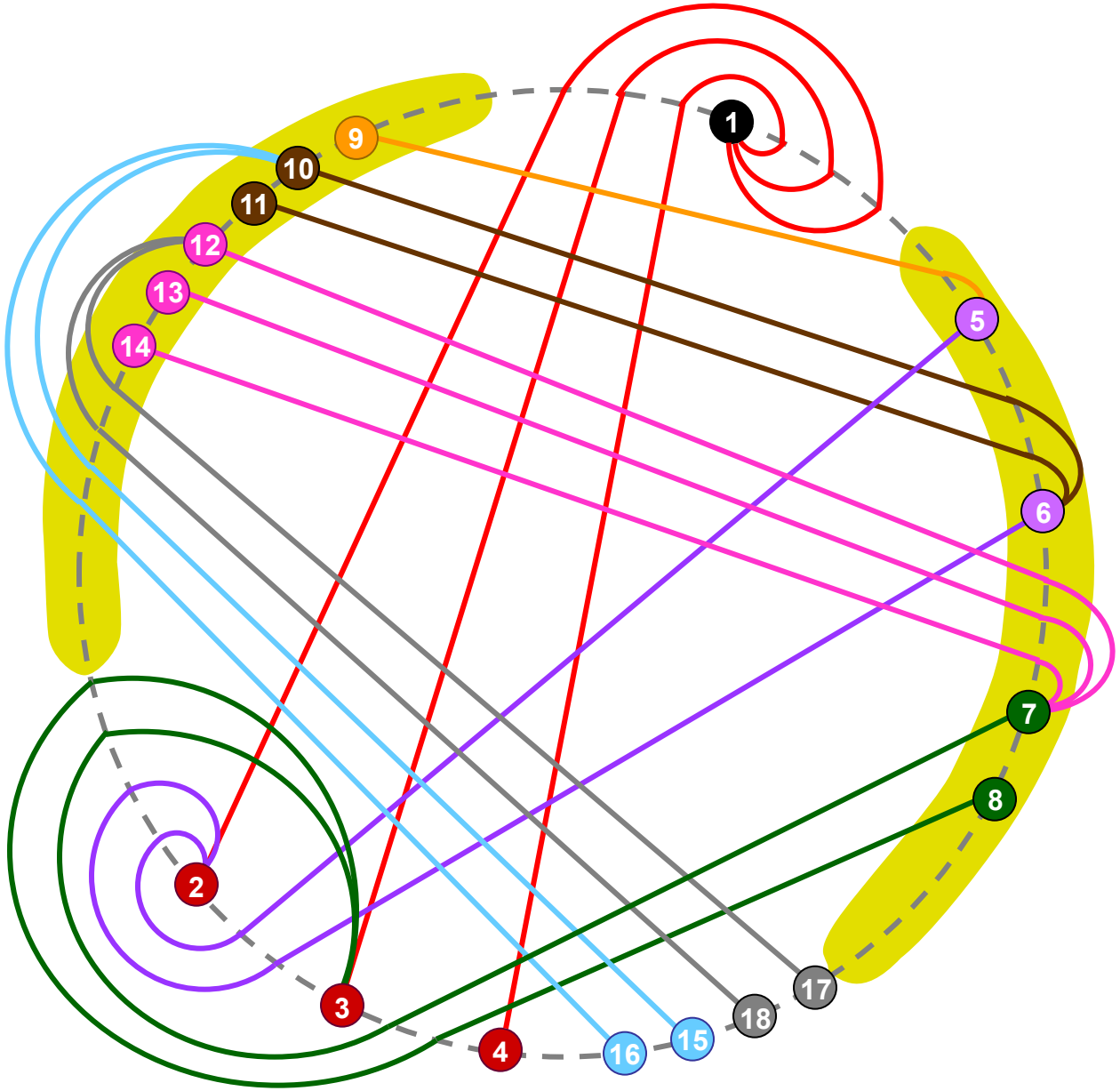
# Making full rainbows



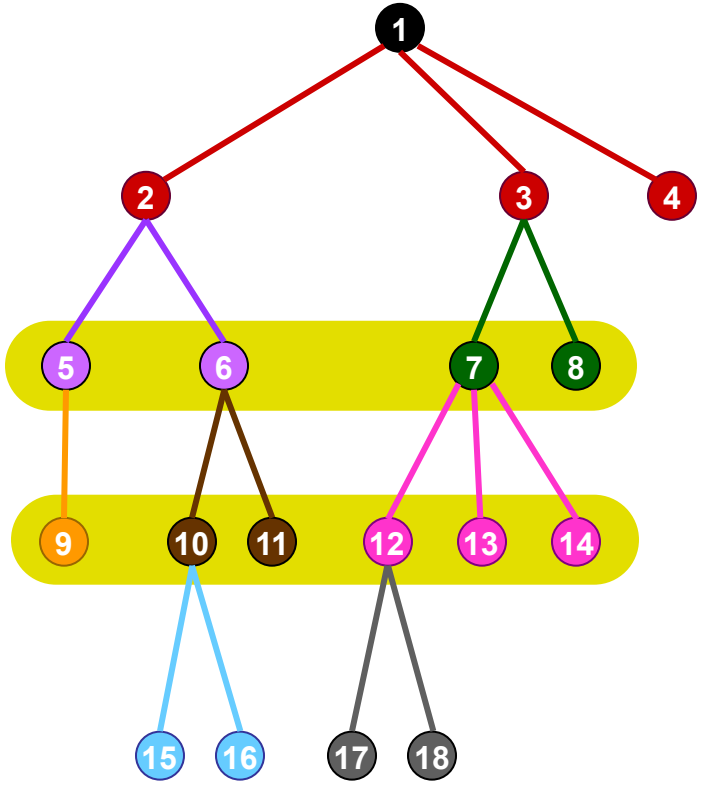



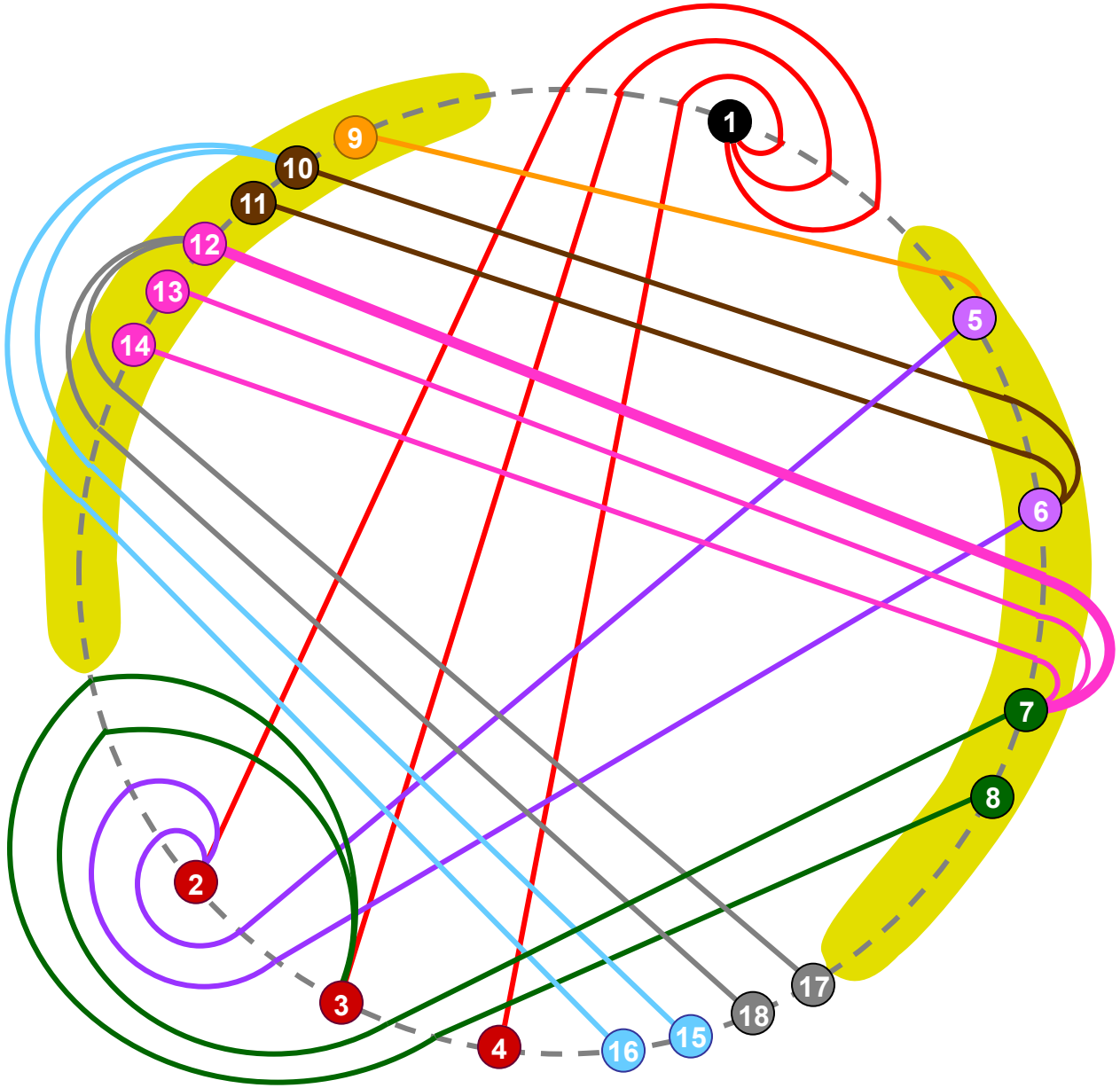
# Making full rainbows



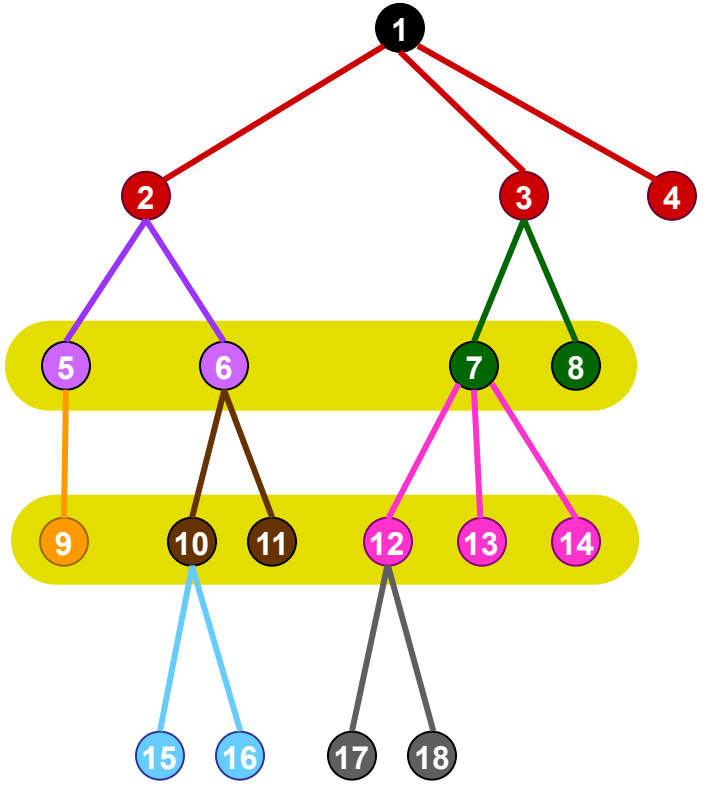



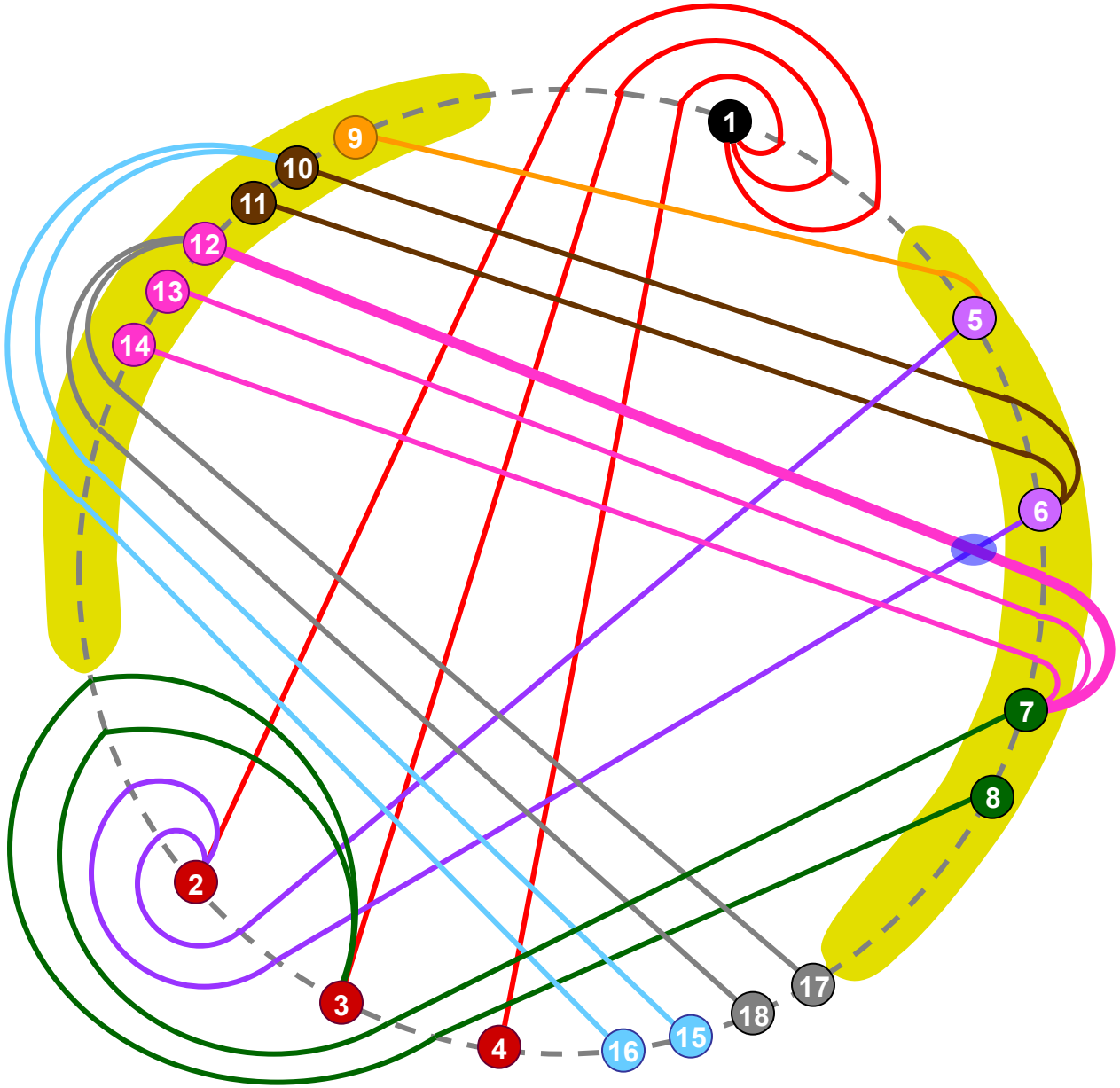
# Making full rainbows



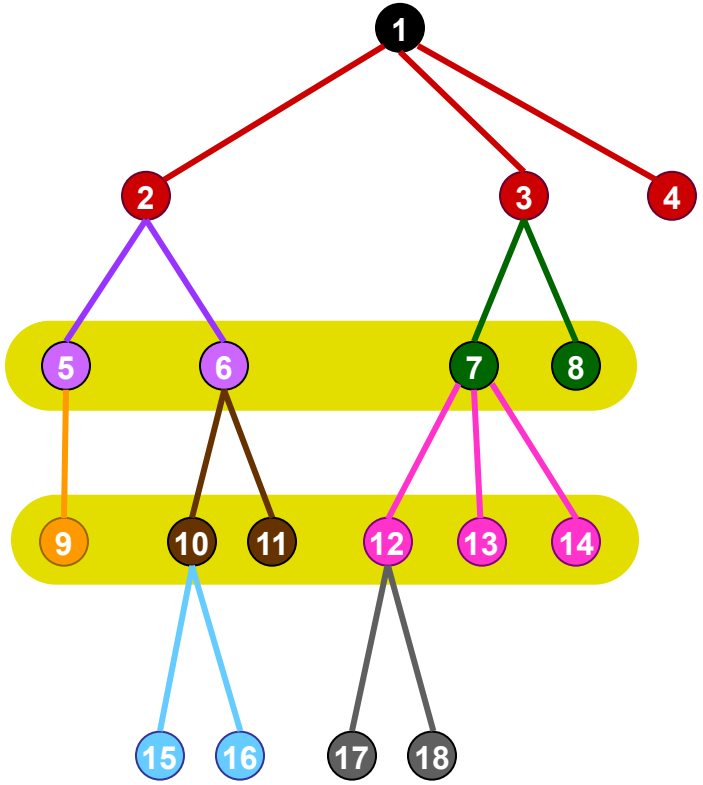
# Making full rainbows

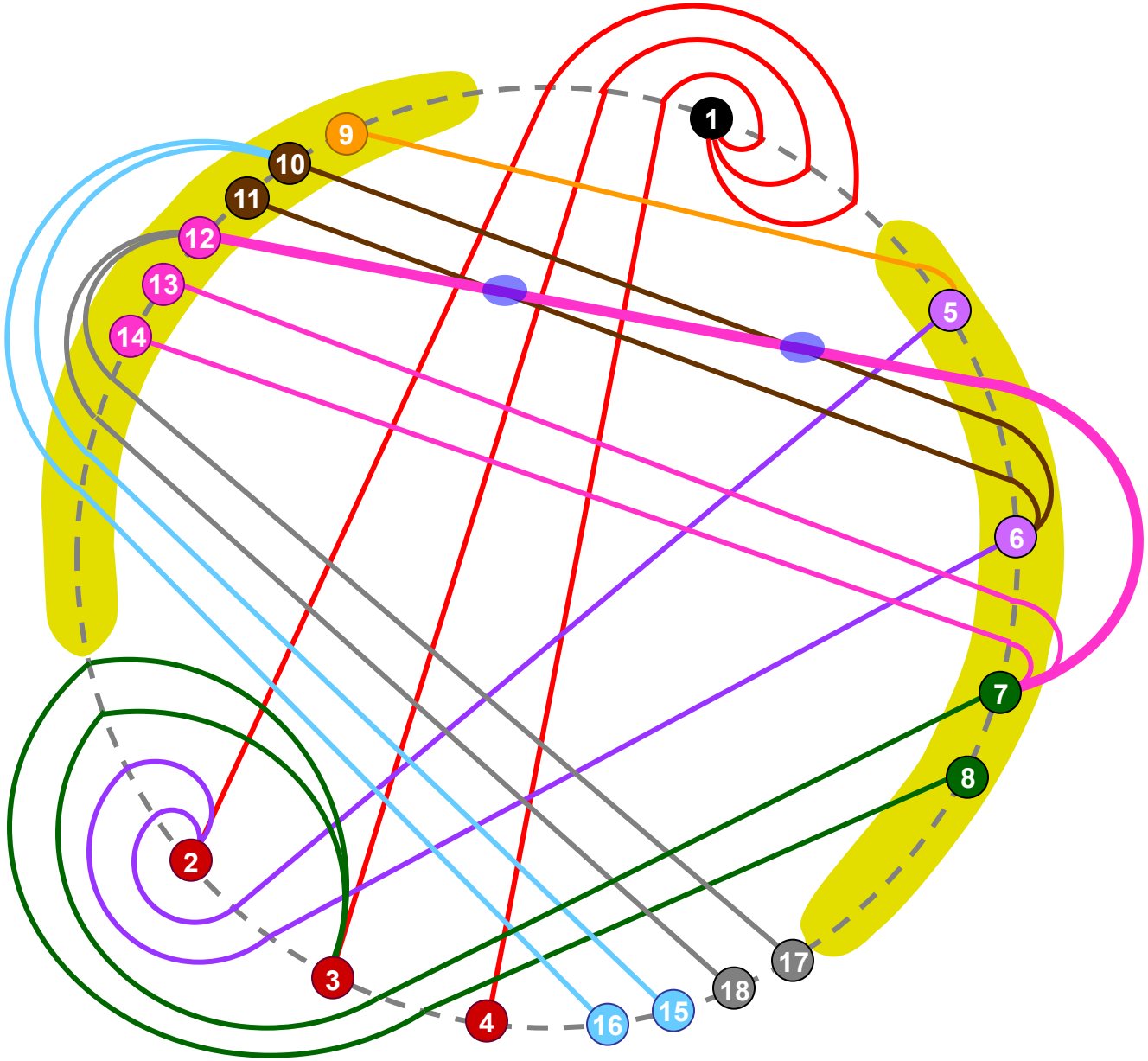


# Making full rainbows

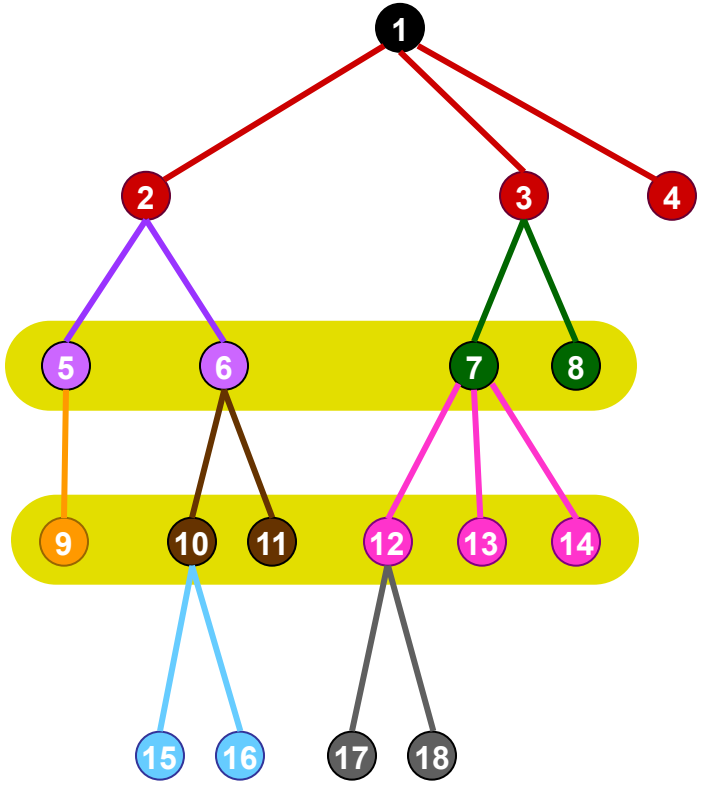
**WARNING**: added 2 unwanted crossings!!!

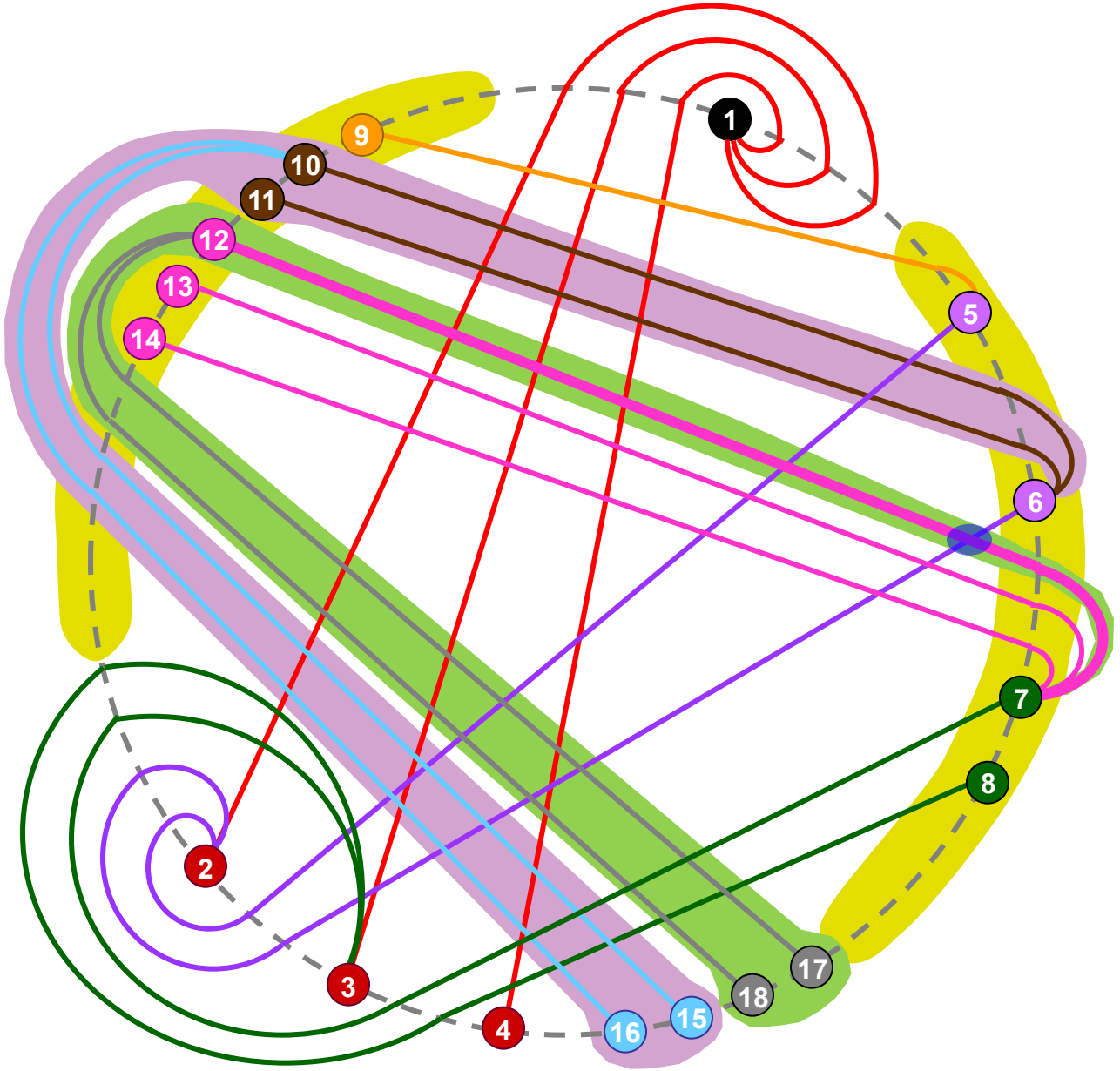


# Making full rainbows

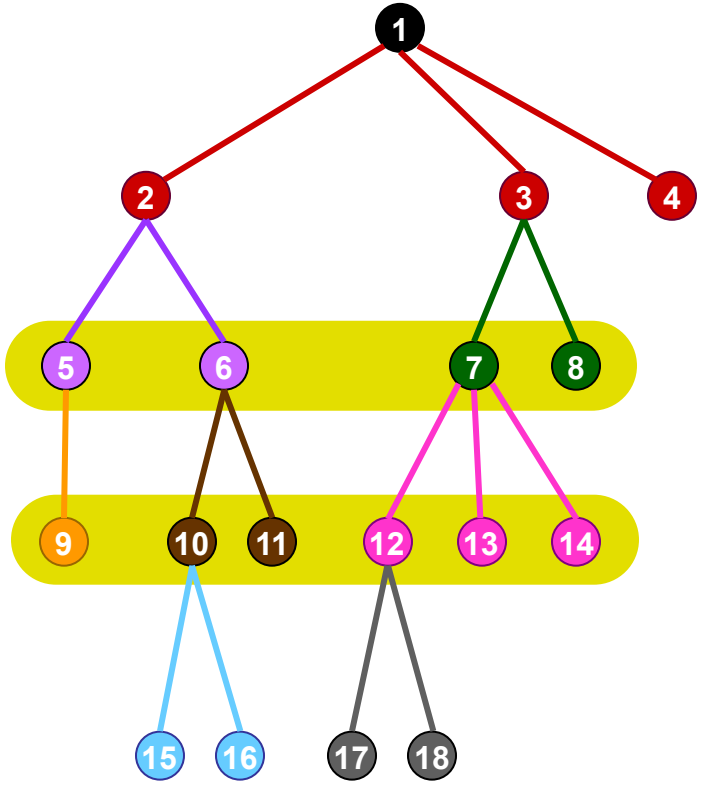
**Fix**: identify subtrees

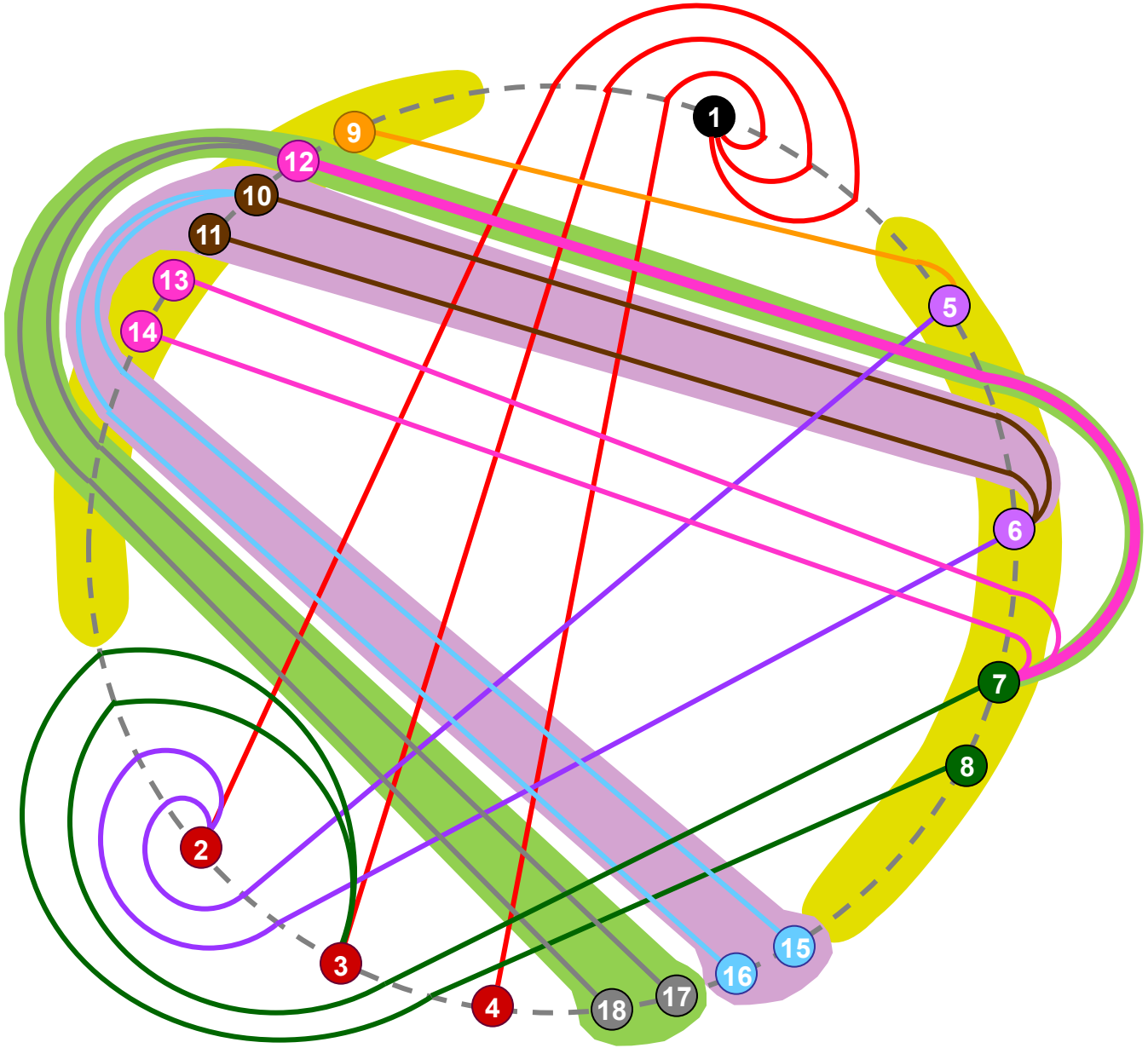


# Making full rainbows

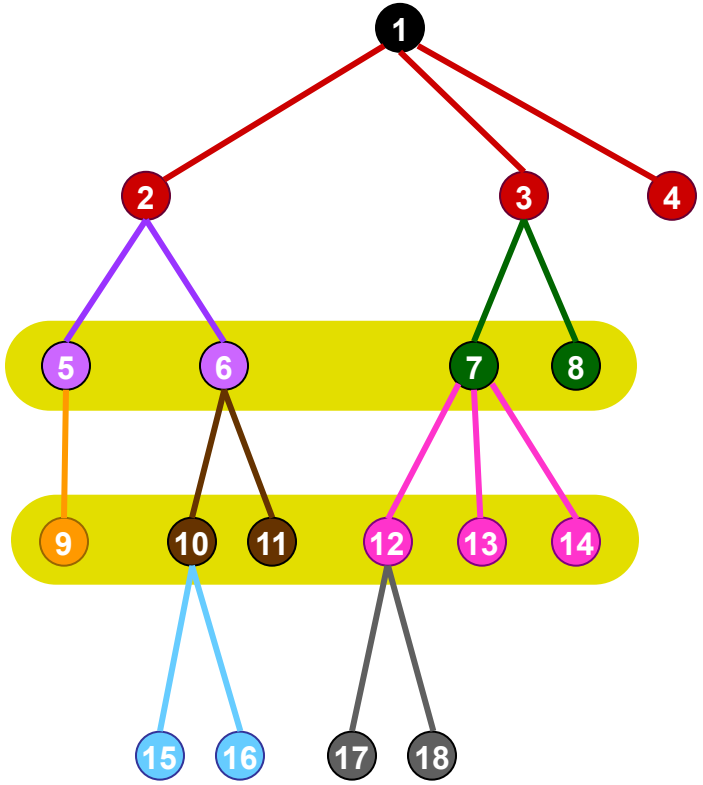
...and swap them...

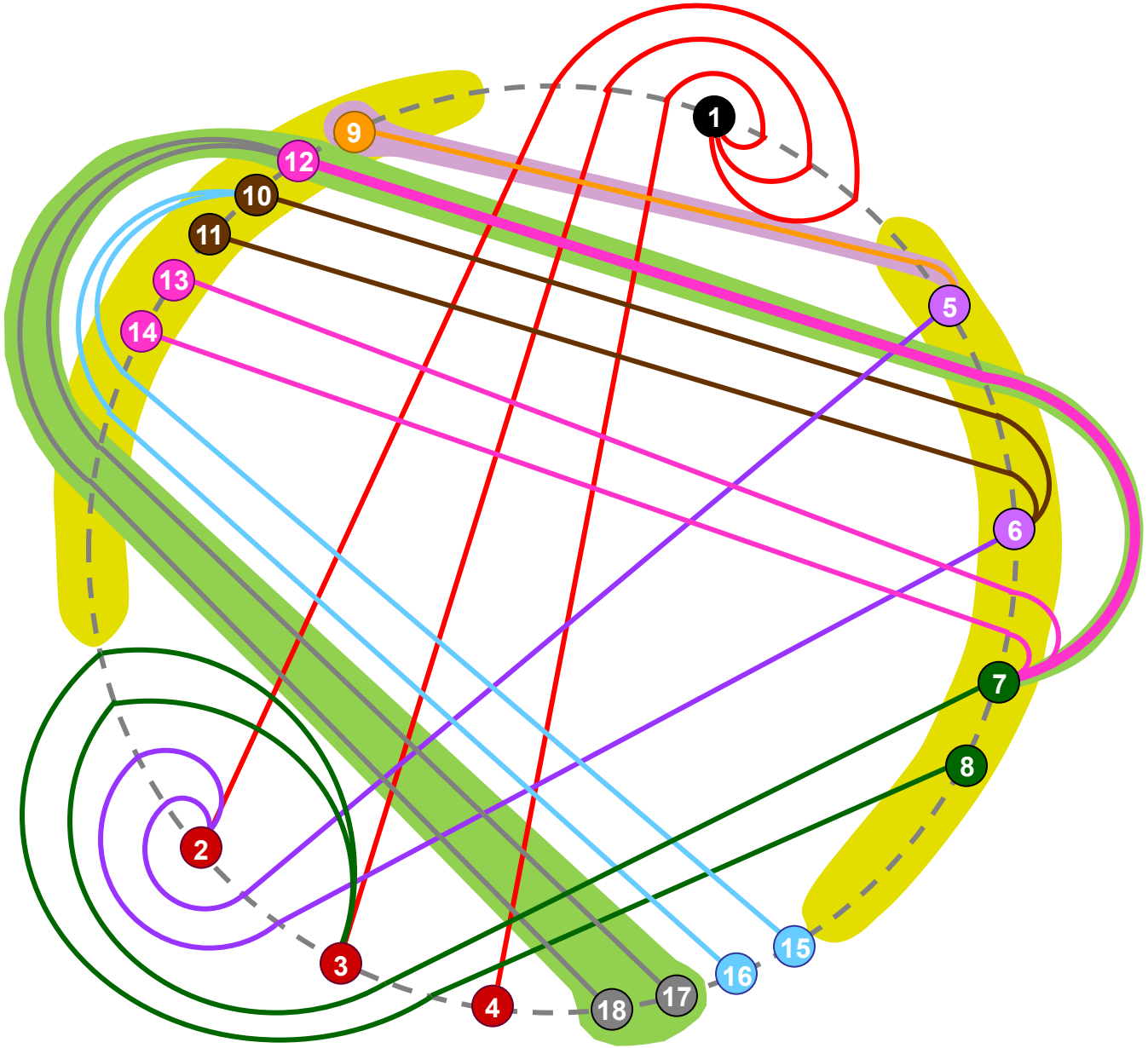


# Making full rainbows

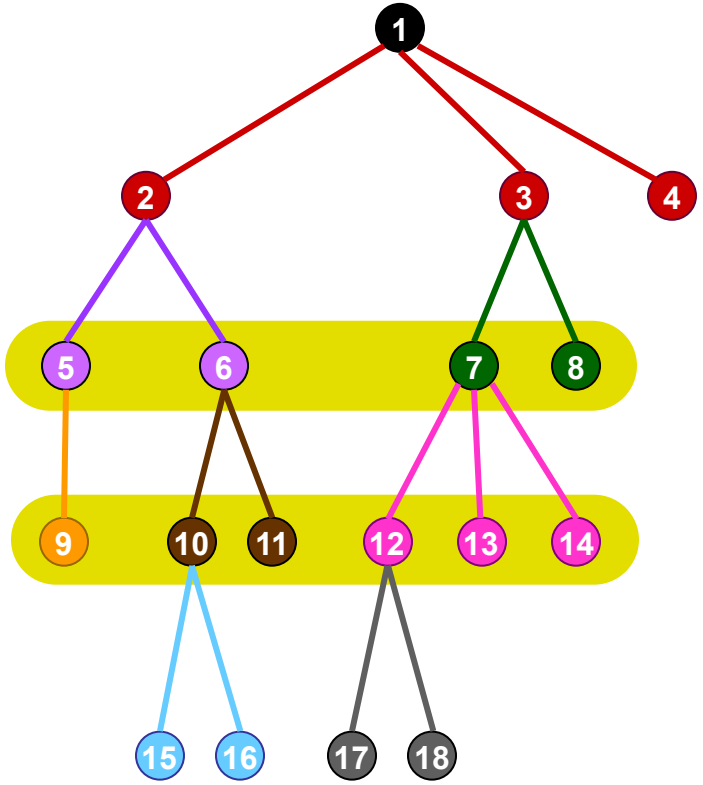
...identify subtrees...

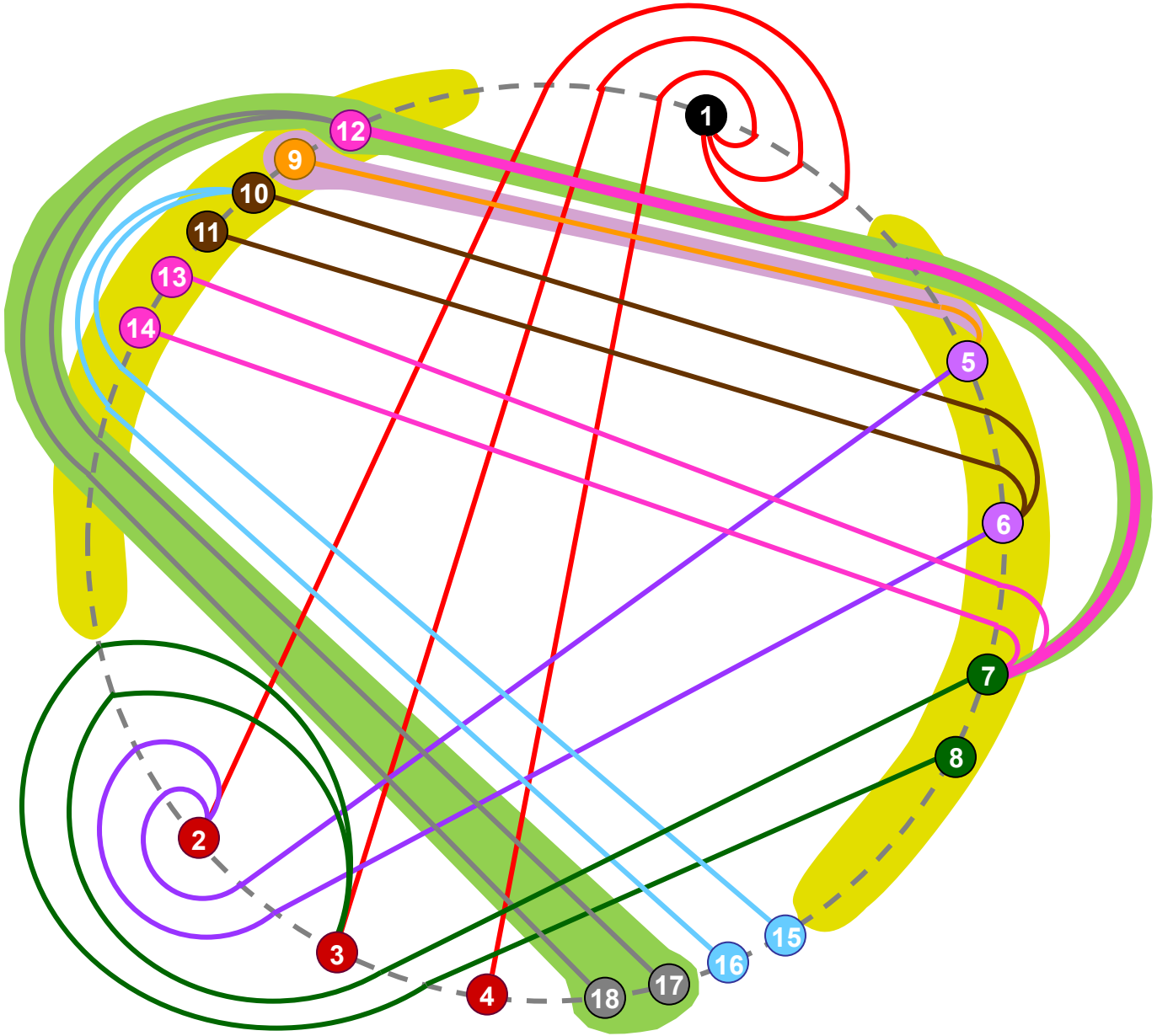


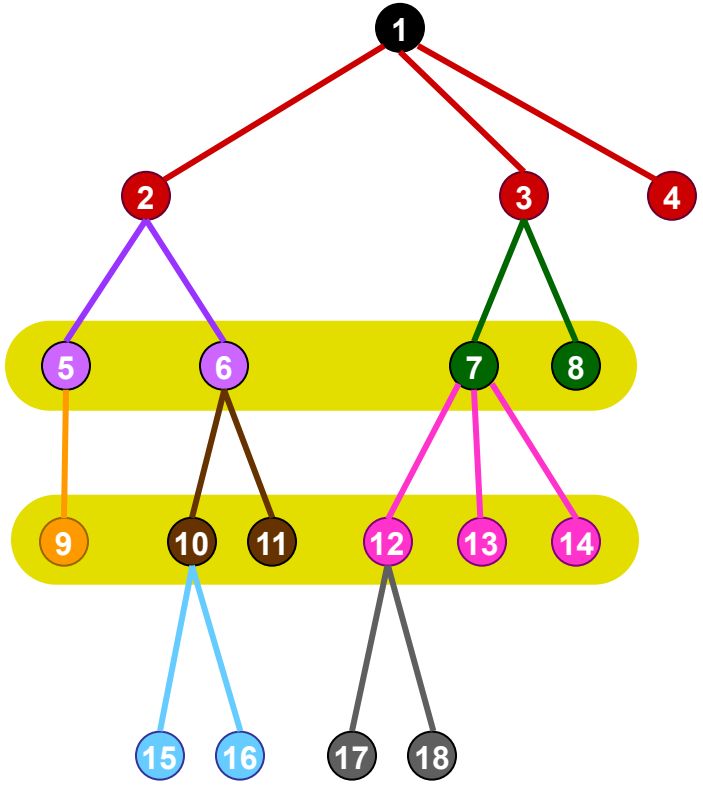
# Making full rainbows

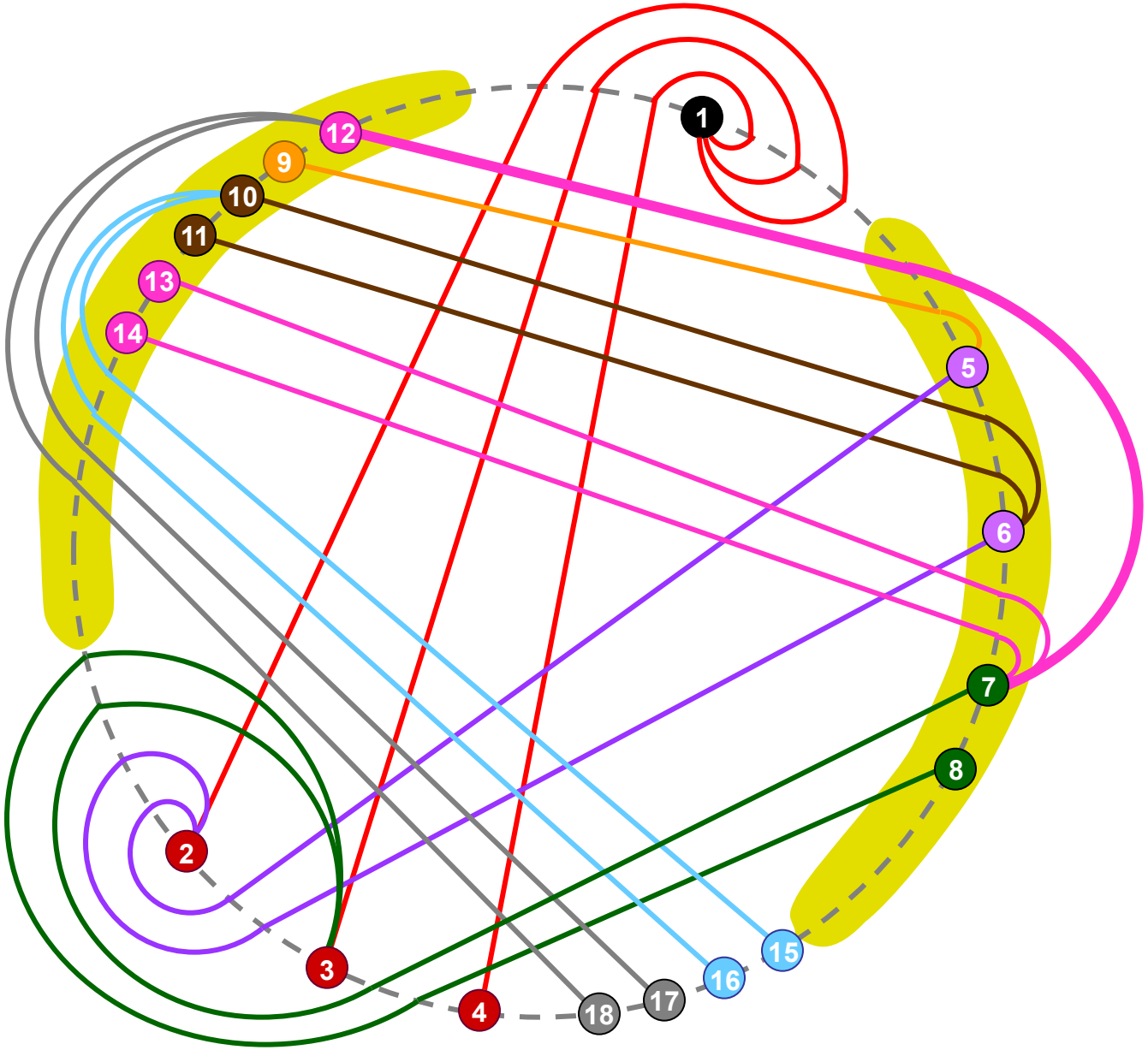
...and swap them...



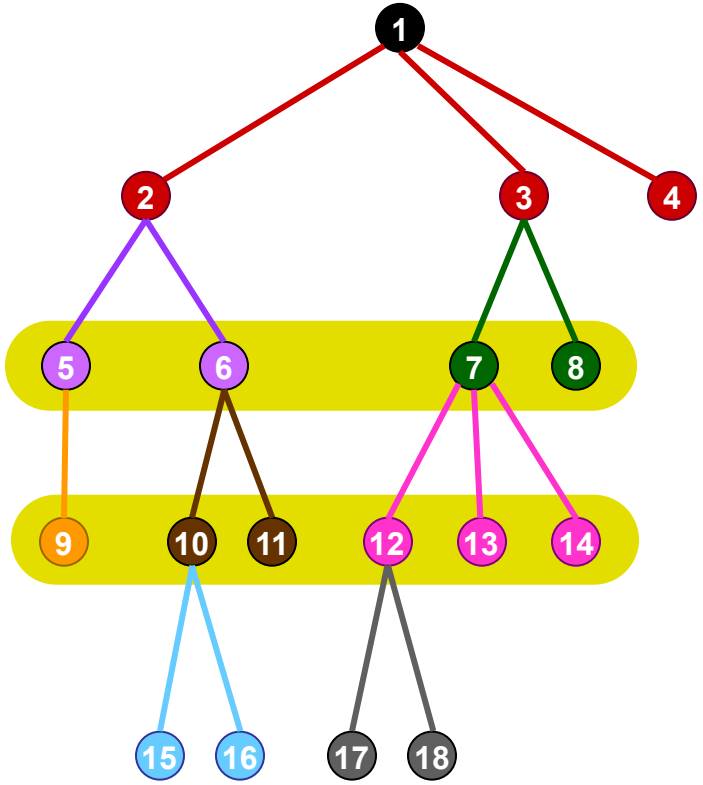



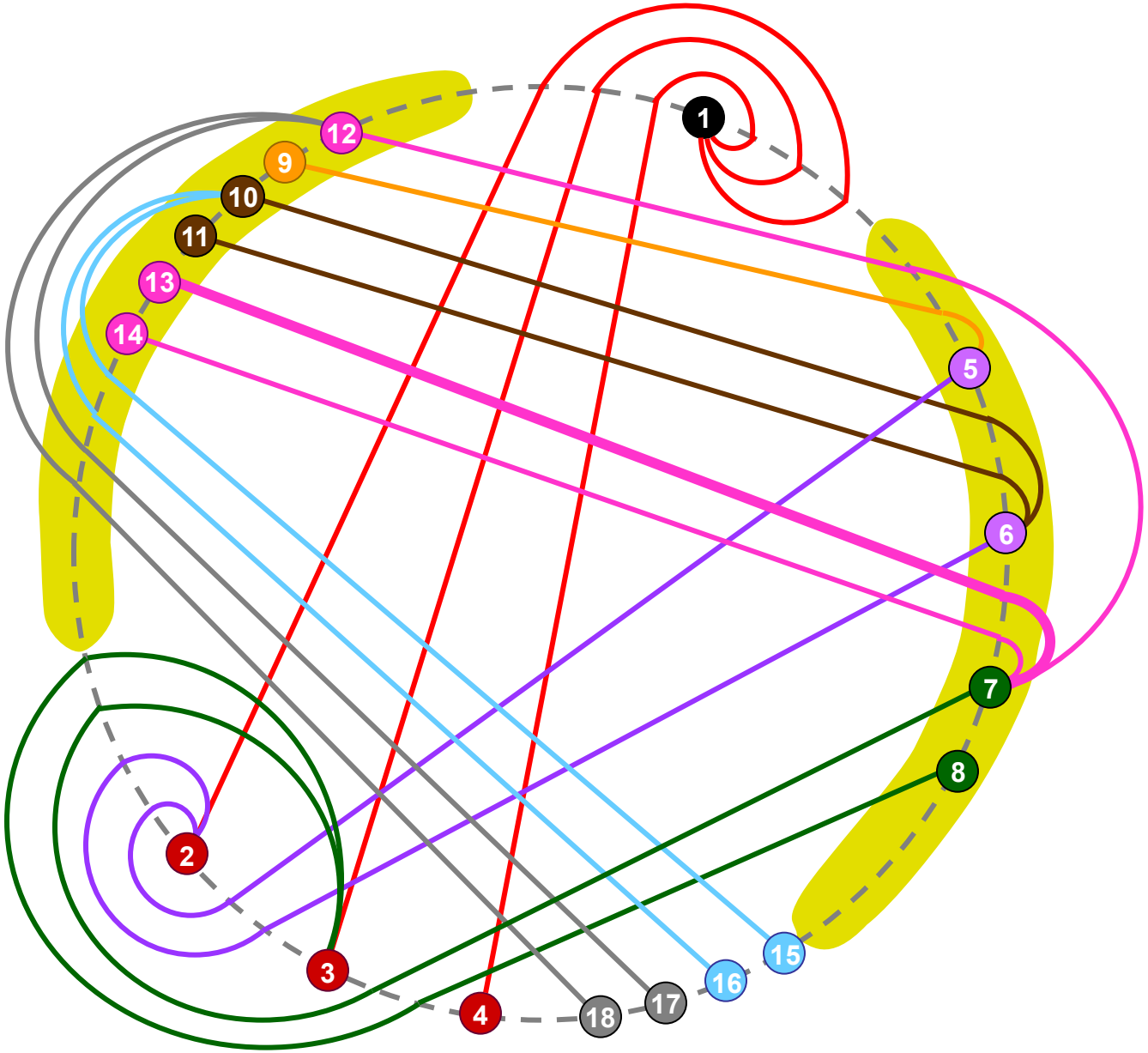
# Making full rainbows



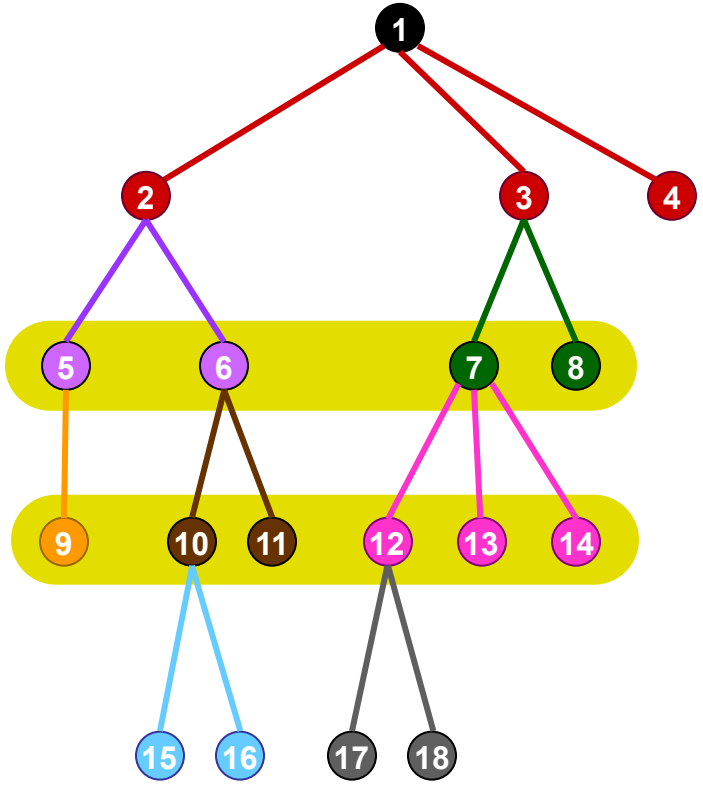



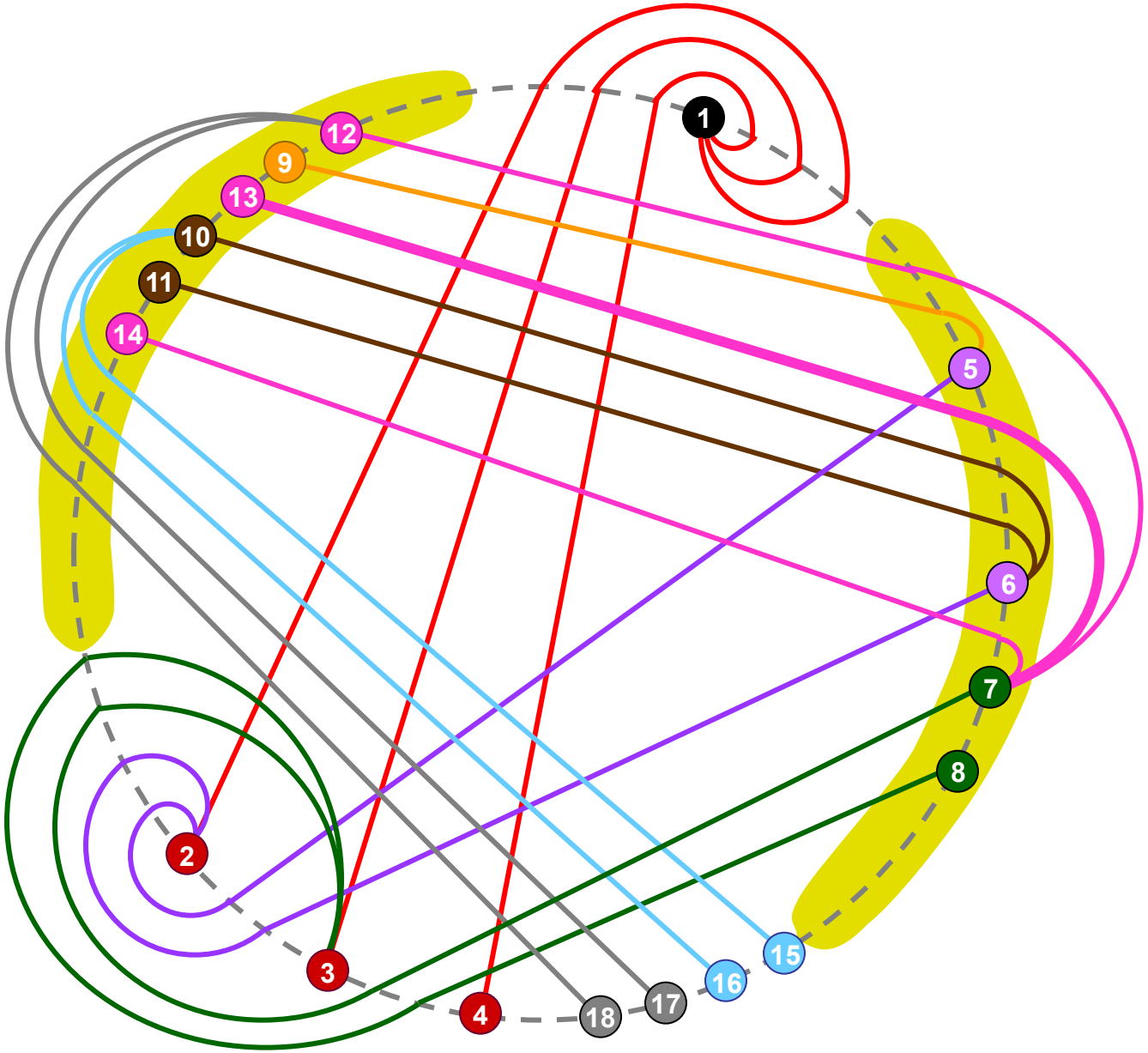
# Making full rainbows



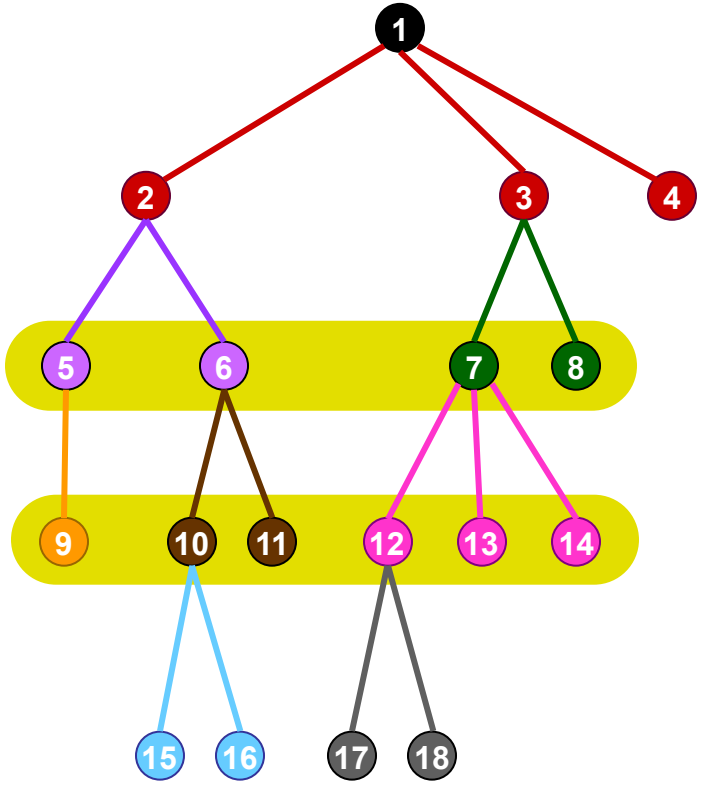



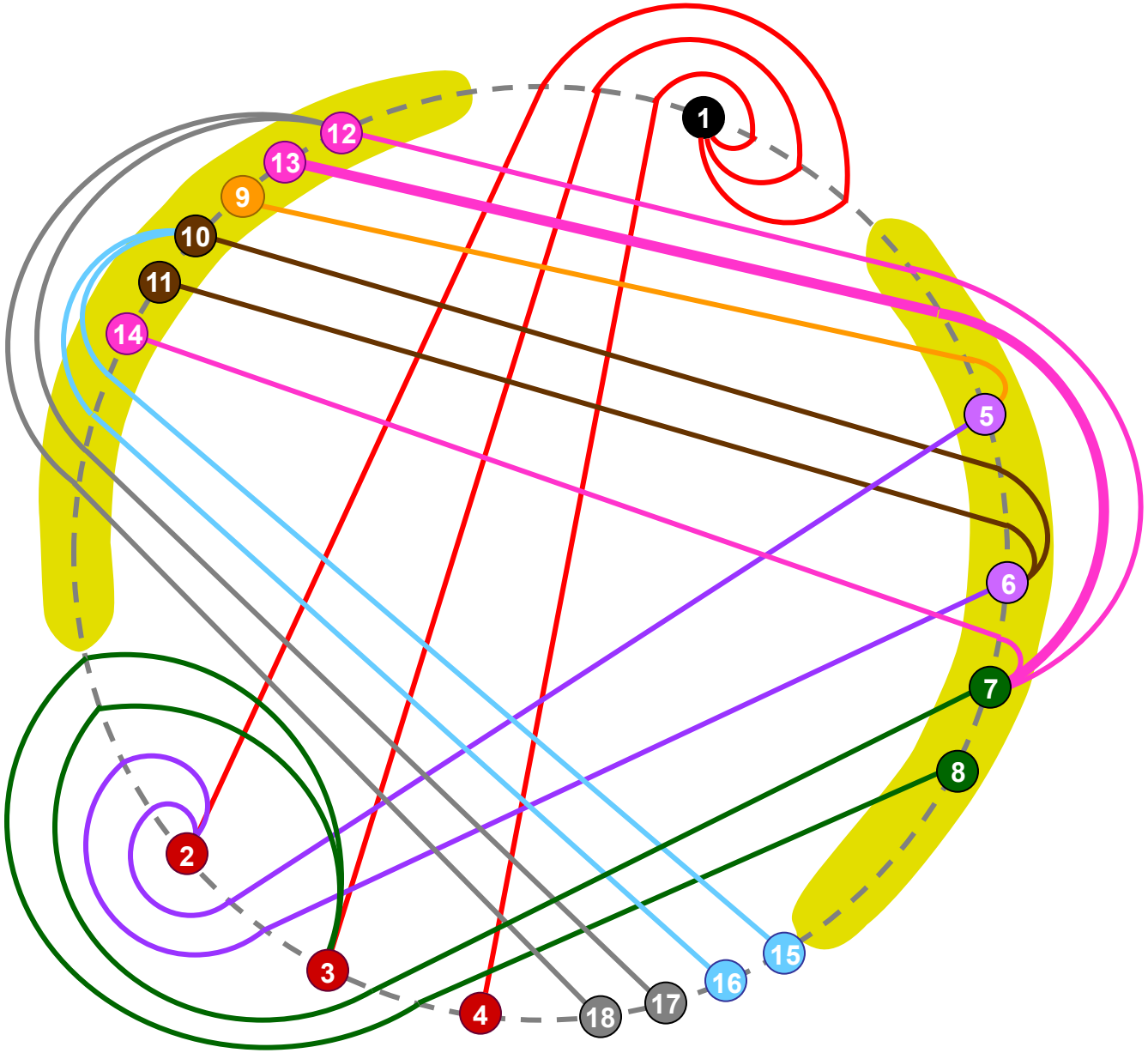
# Making full rainbows



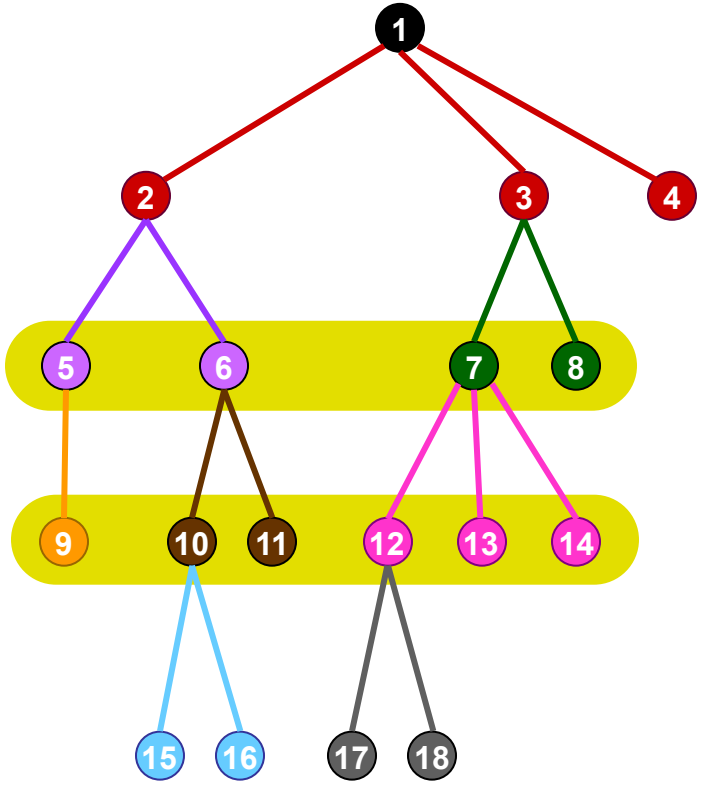



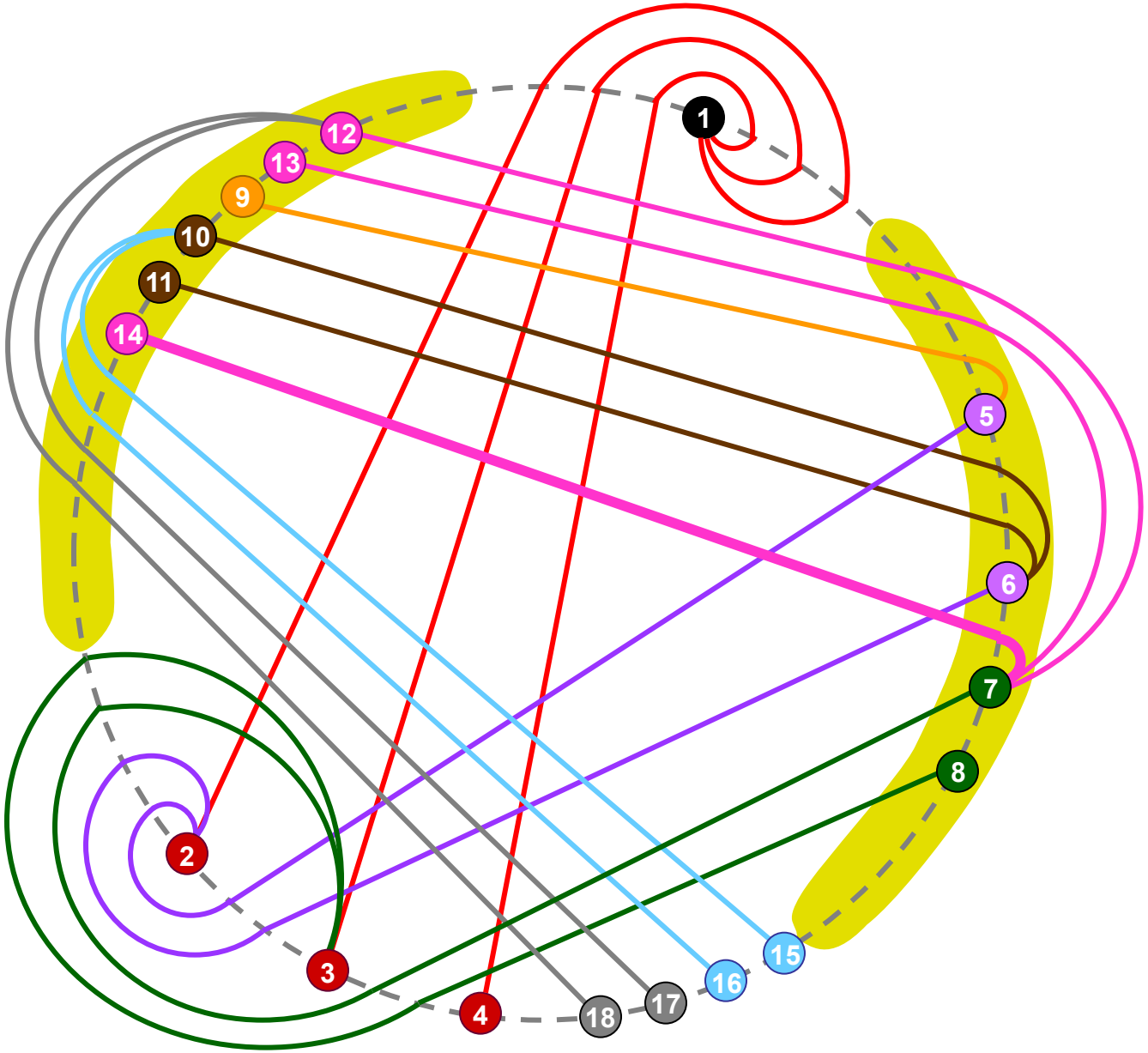
# Making full rainbows



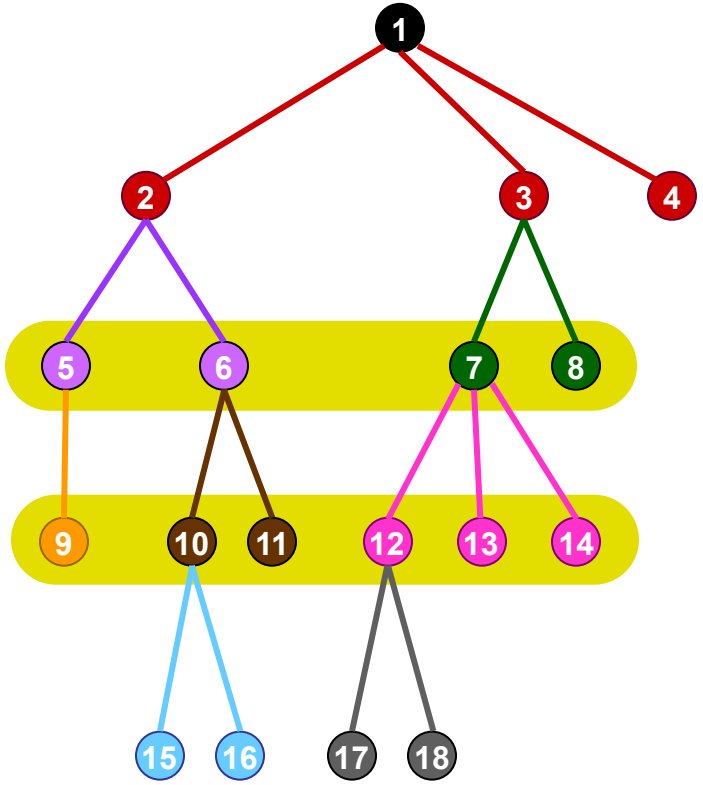



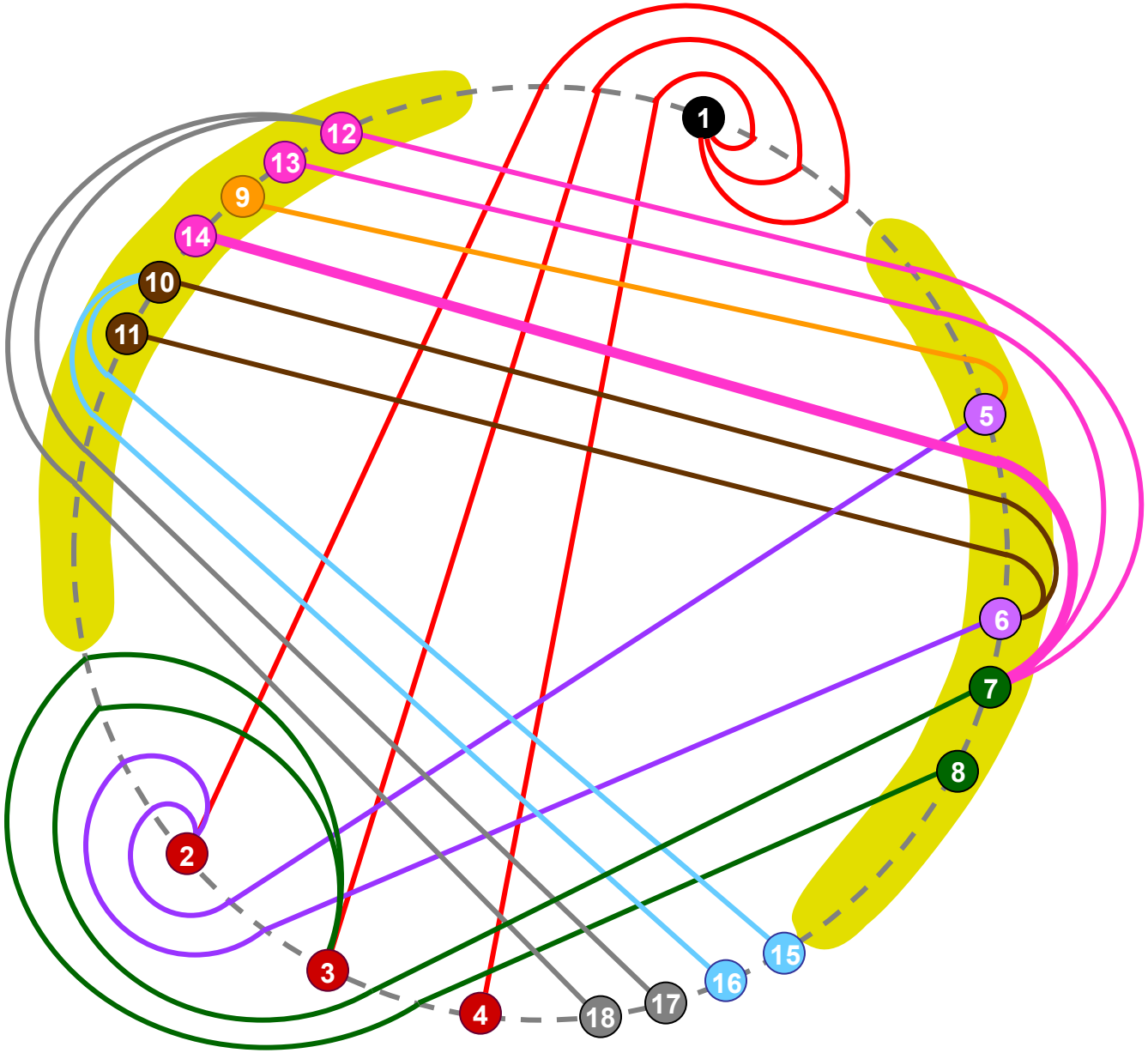
# Making full rainbows



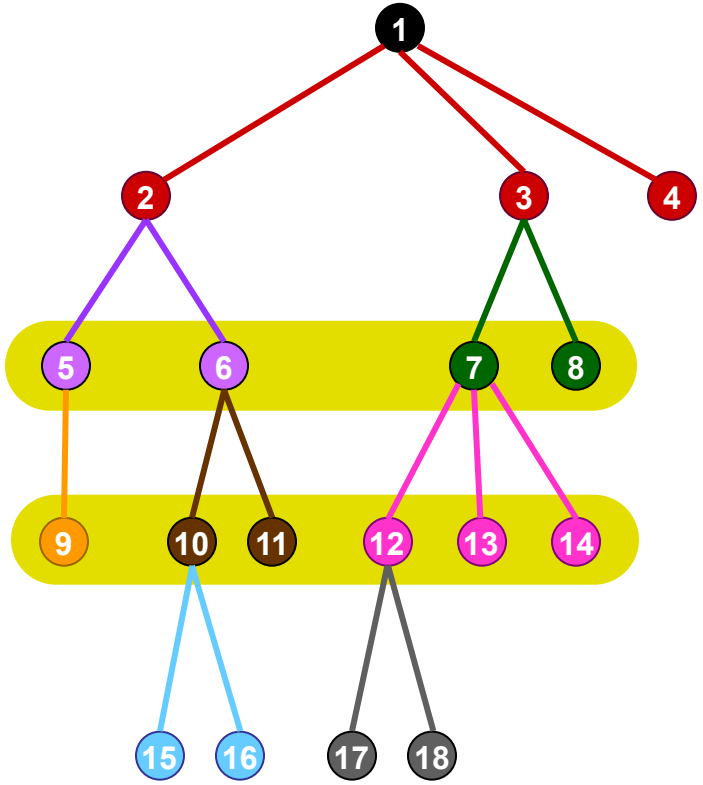



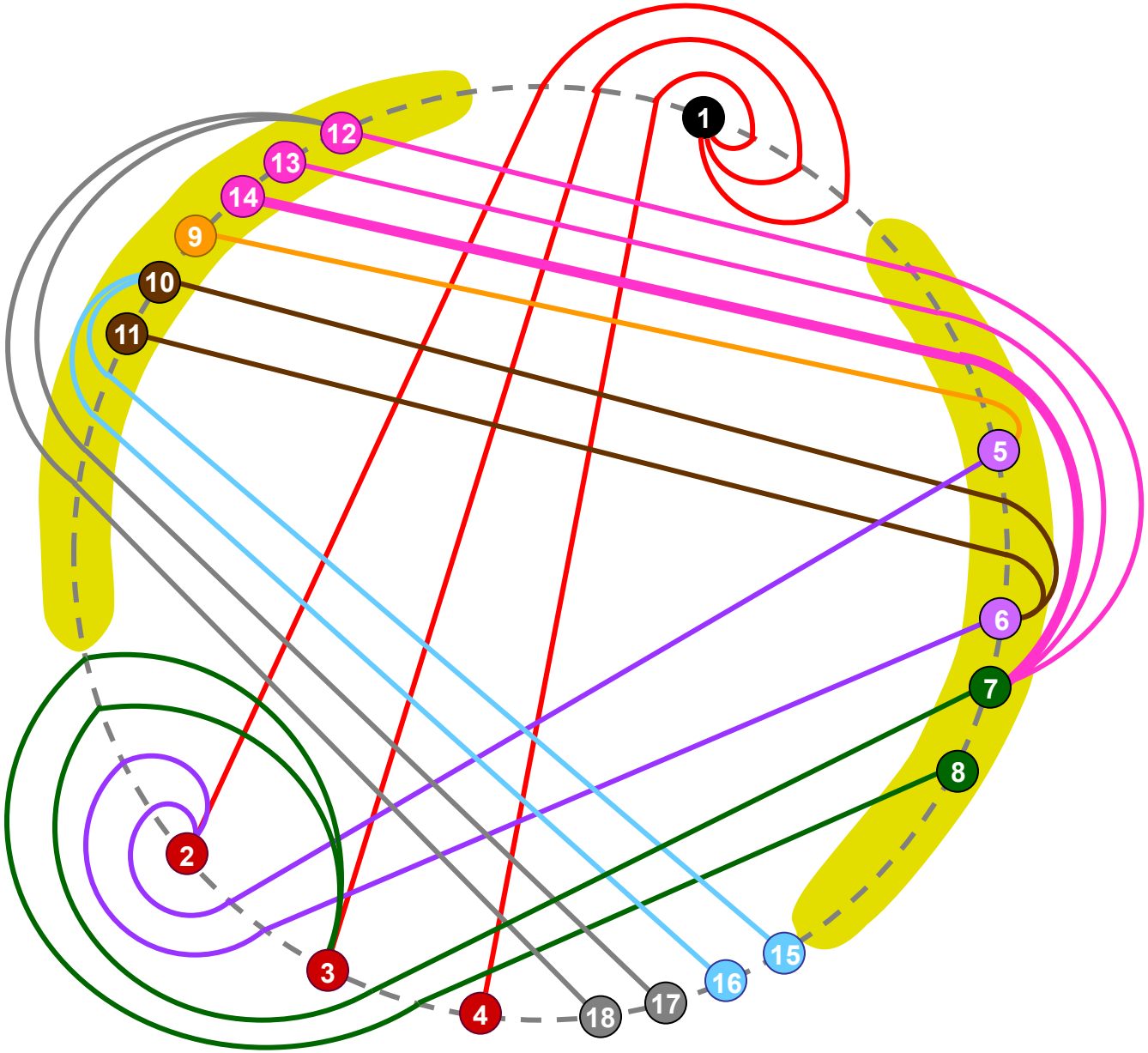
# Making full rainbows



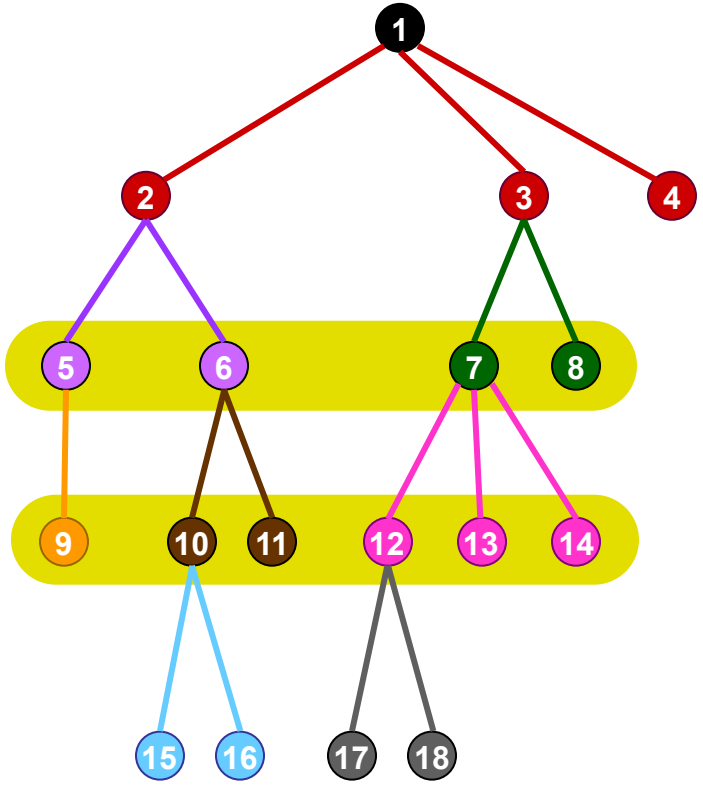



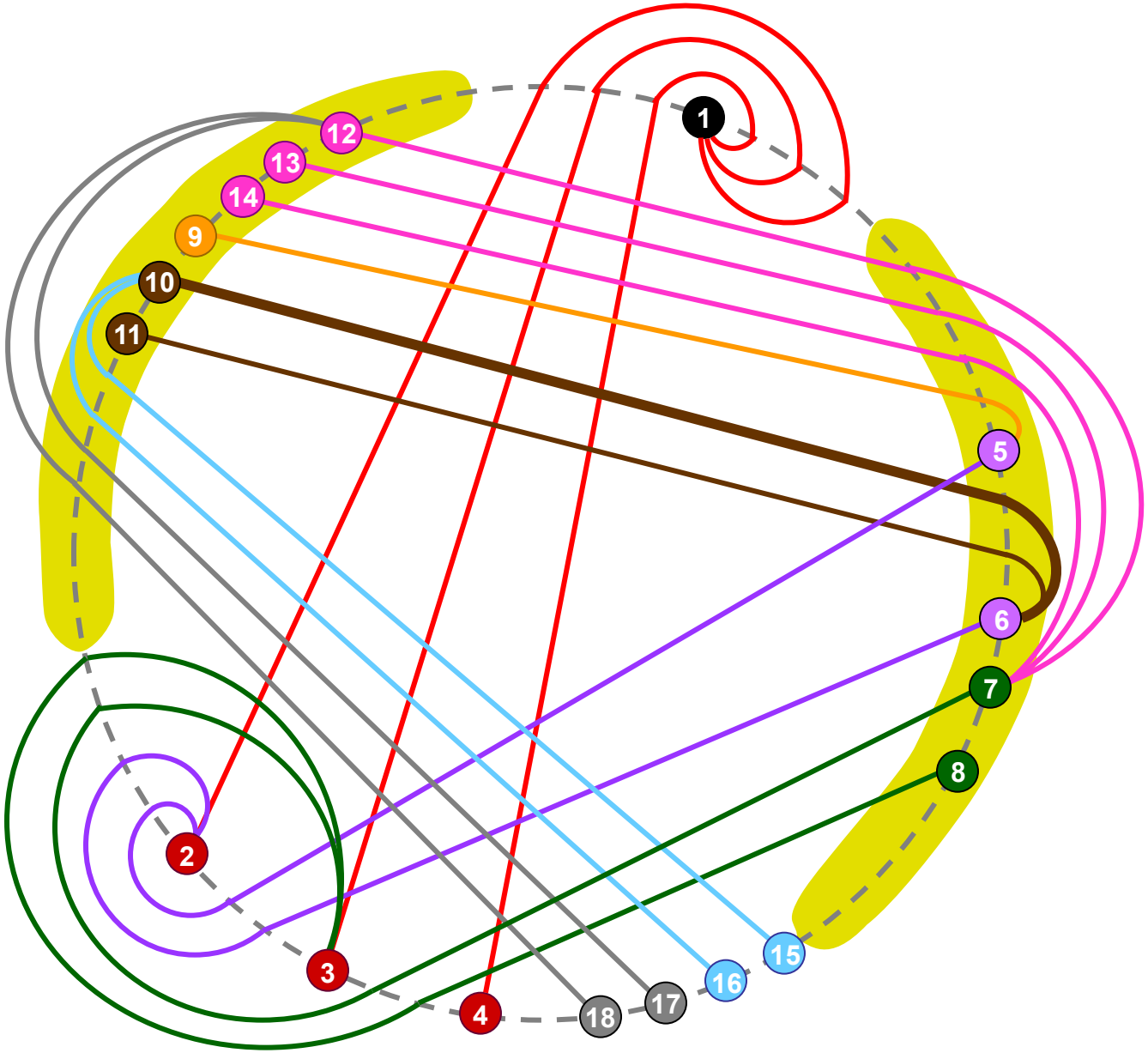
# Making full rainbows



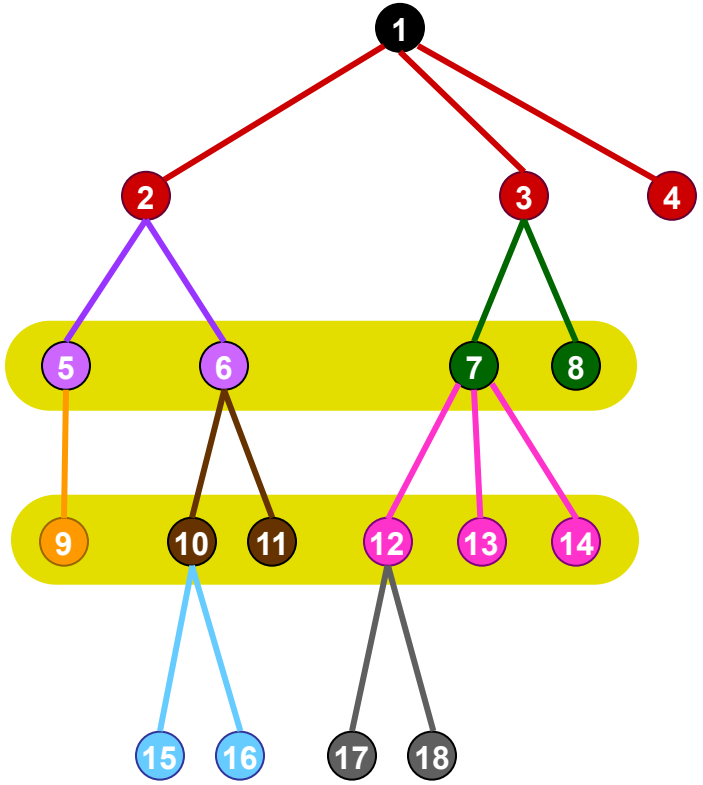



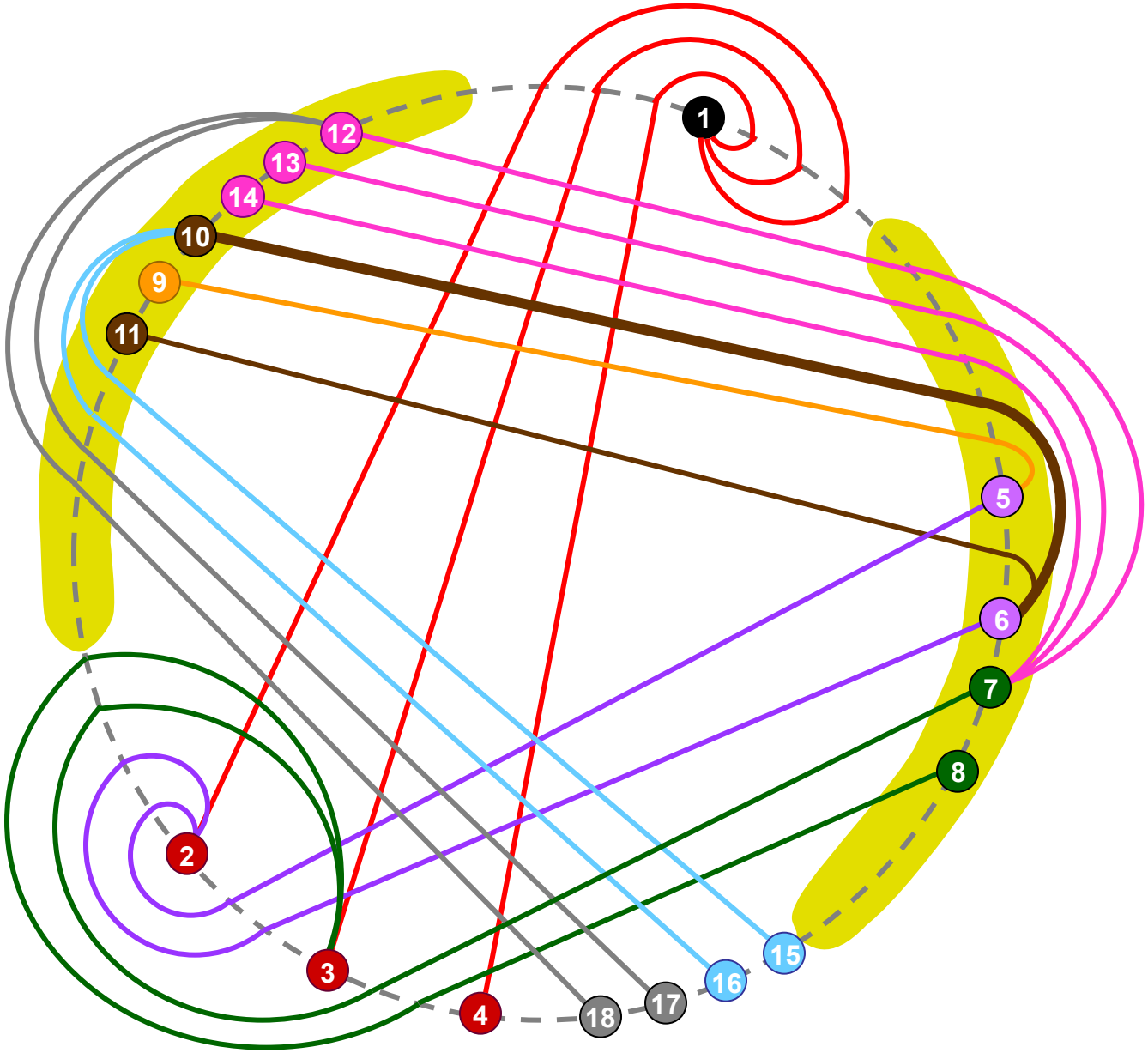
# Making full rainbows



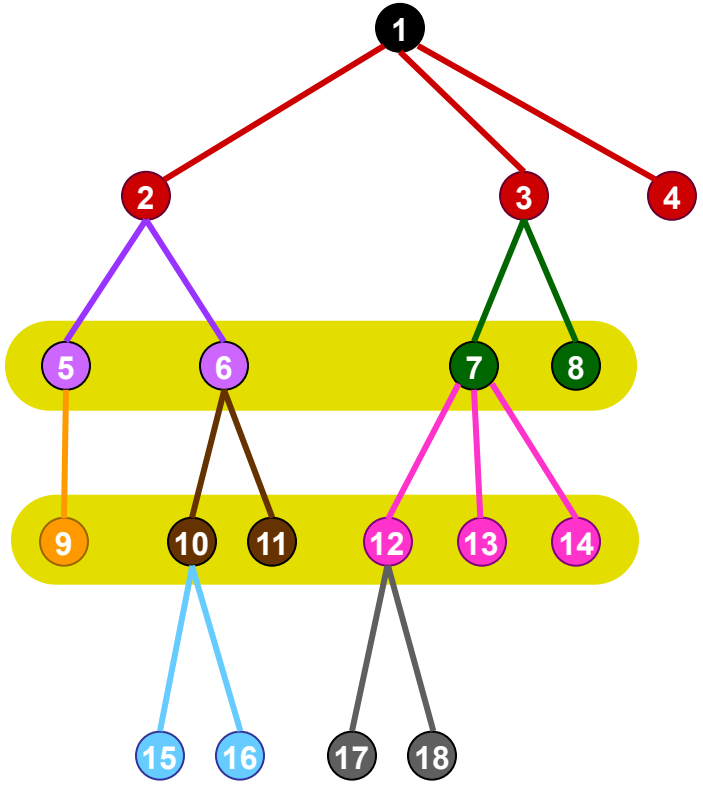



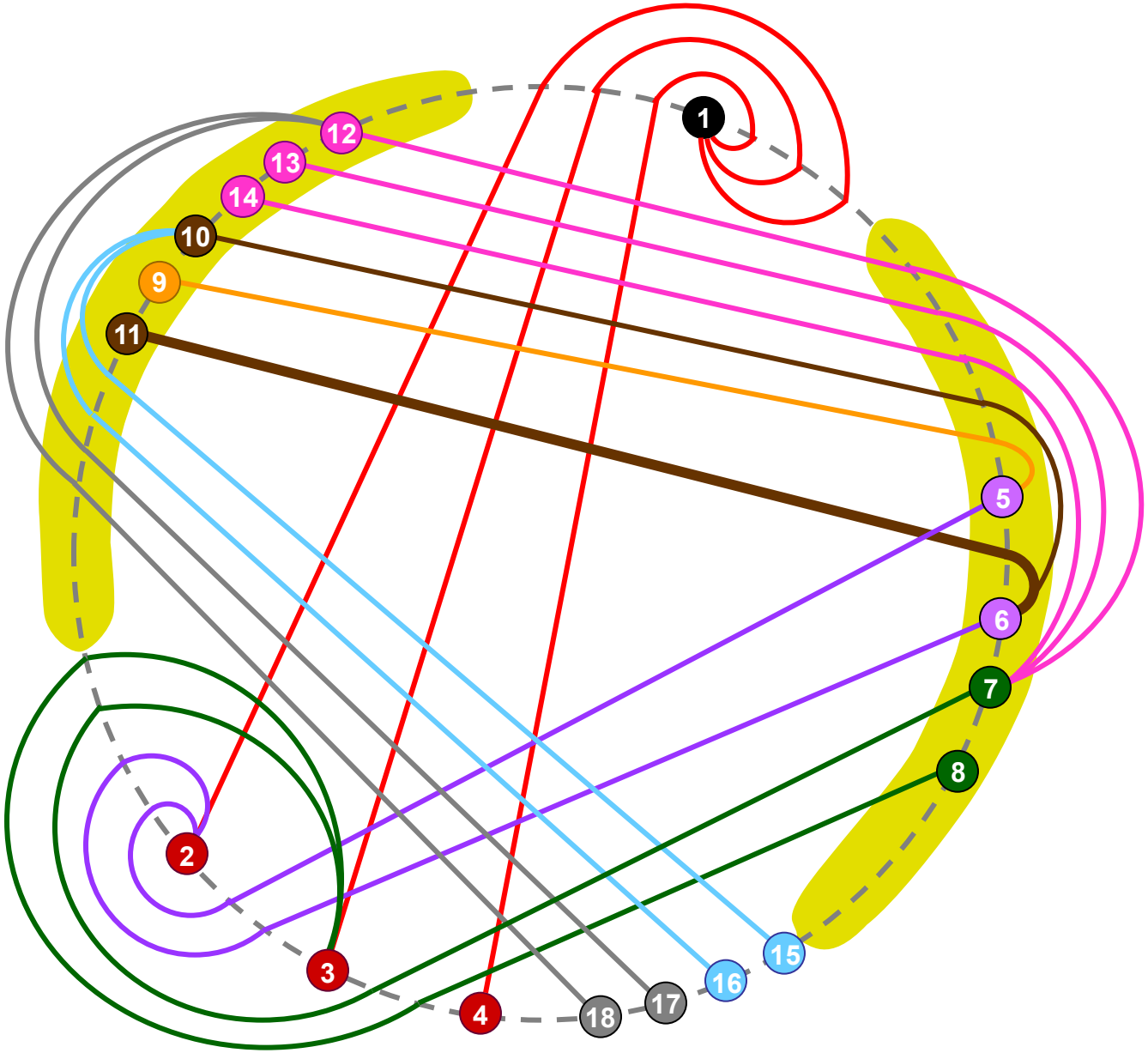
# Making full rainbows



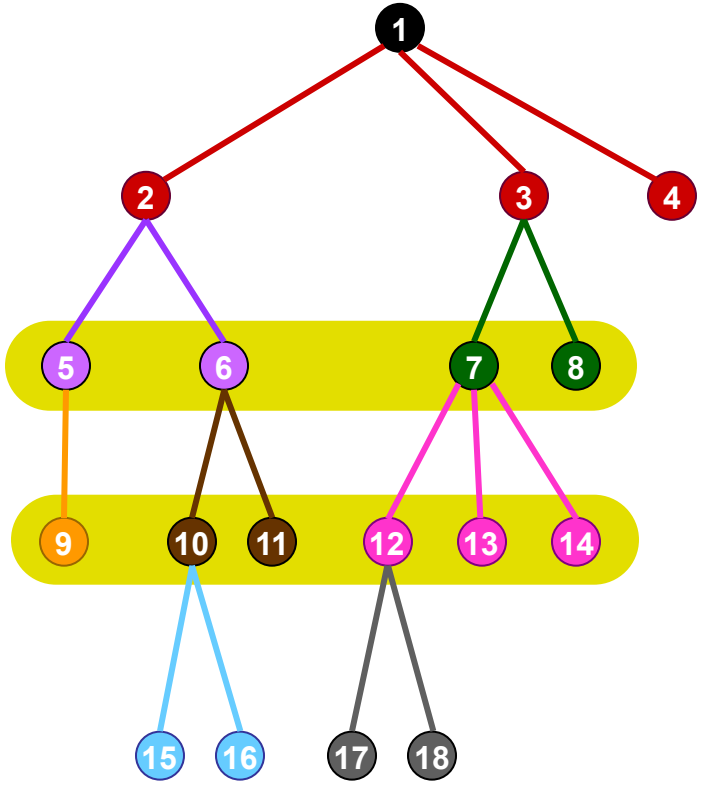



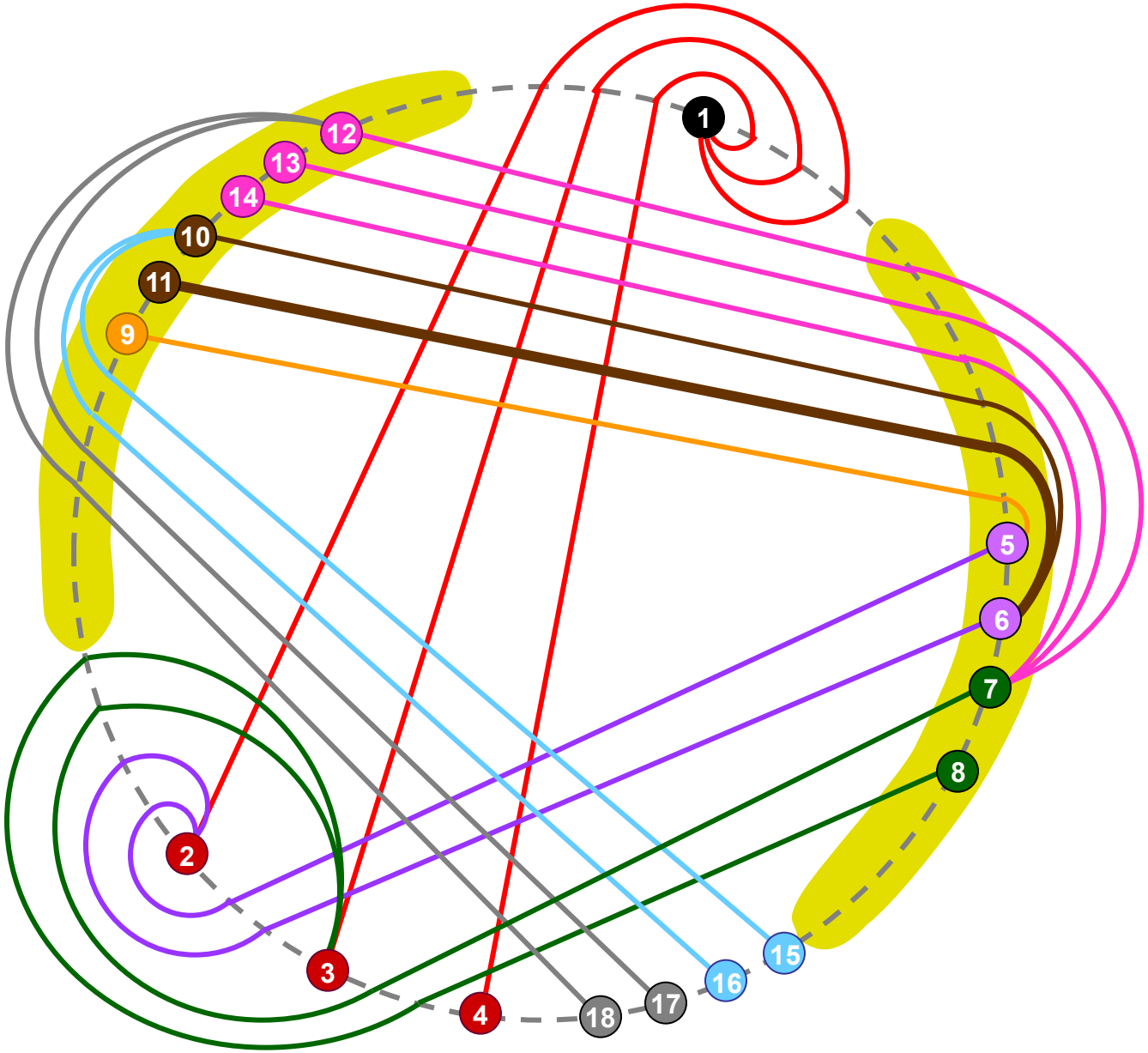
# Making full rainbows



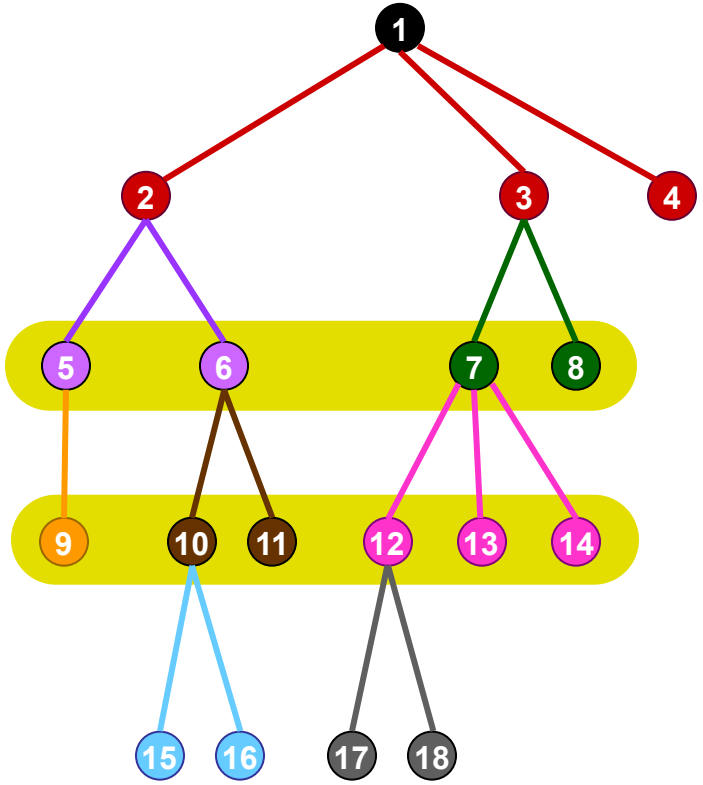



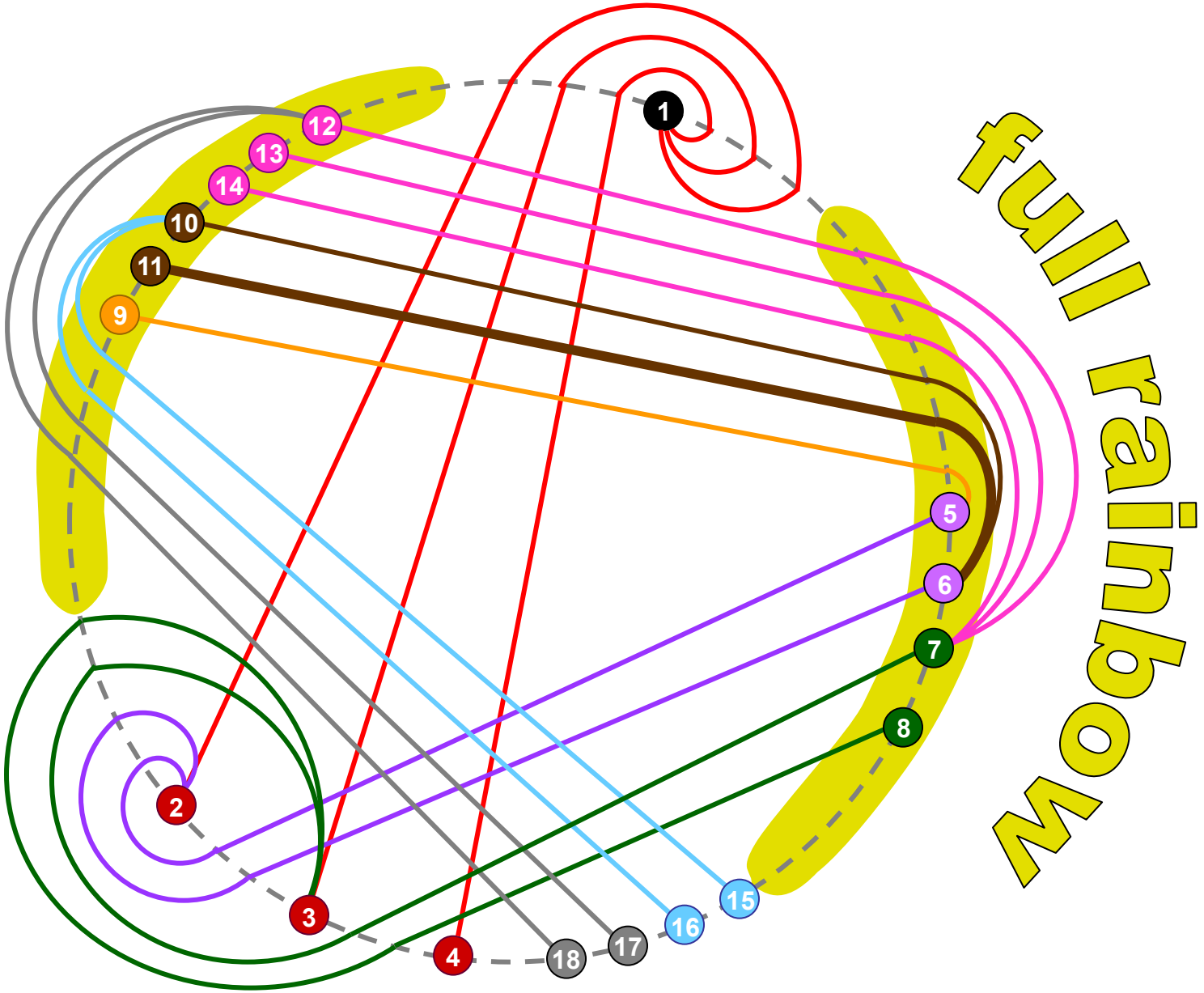
# Making full rainbows



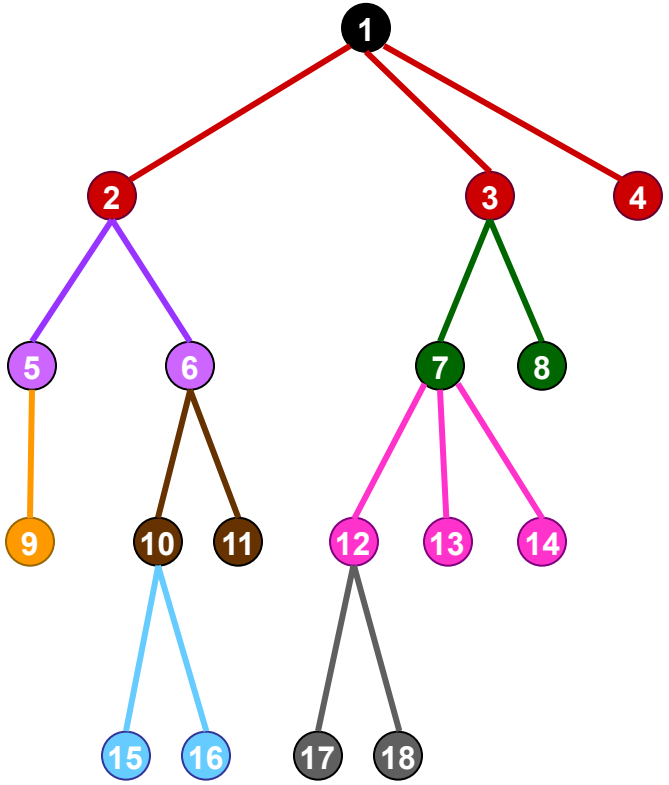



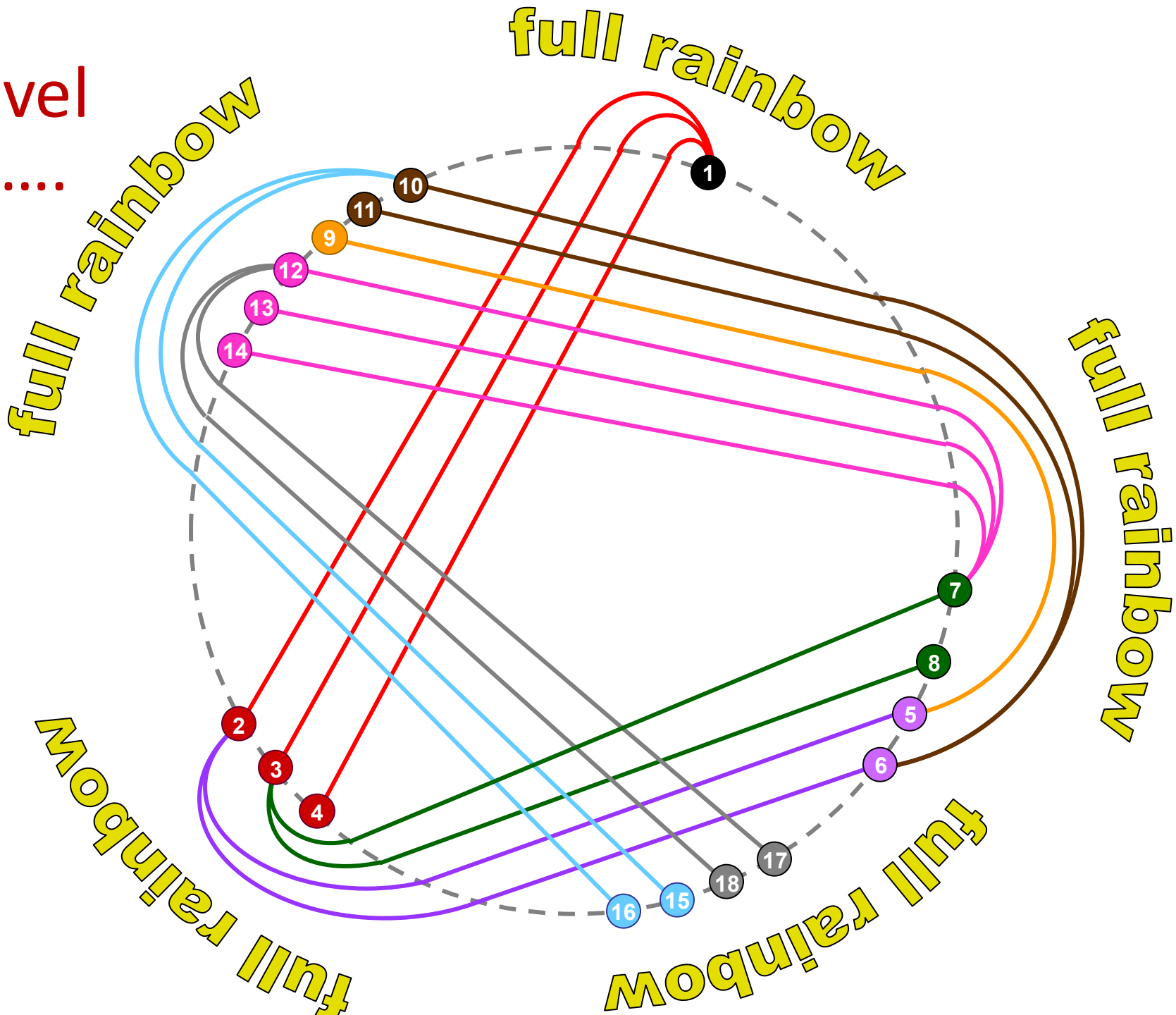
# Making full rainbows



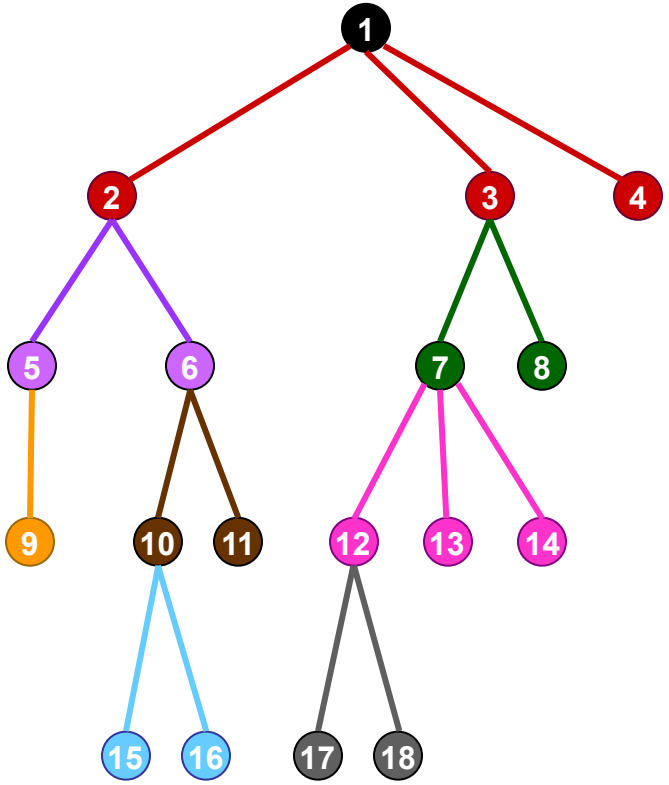



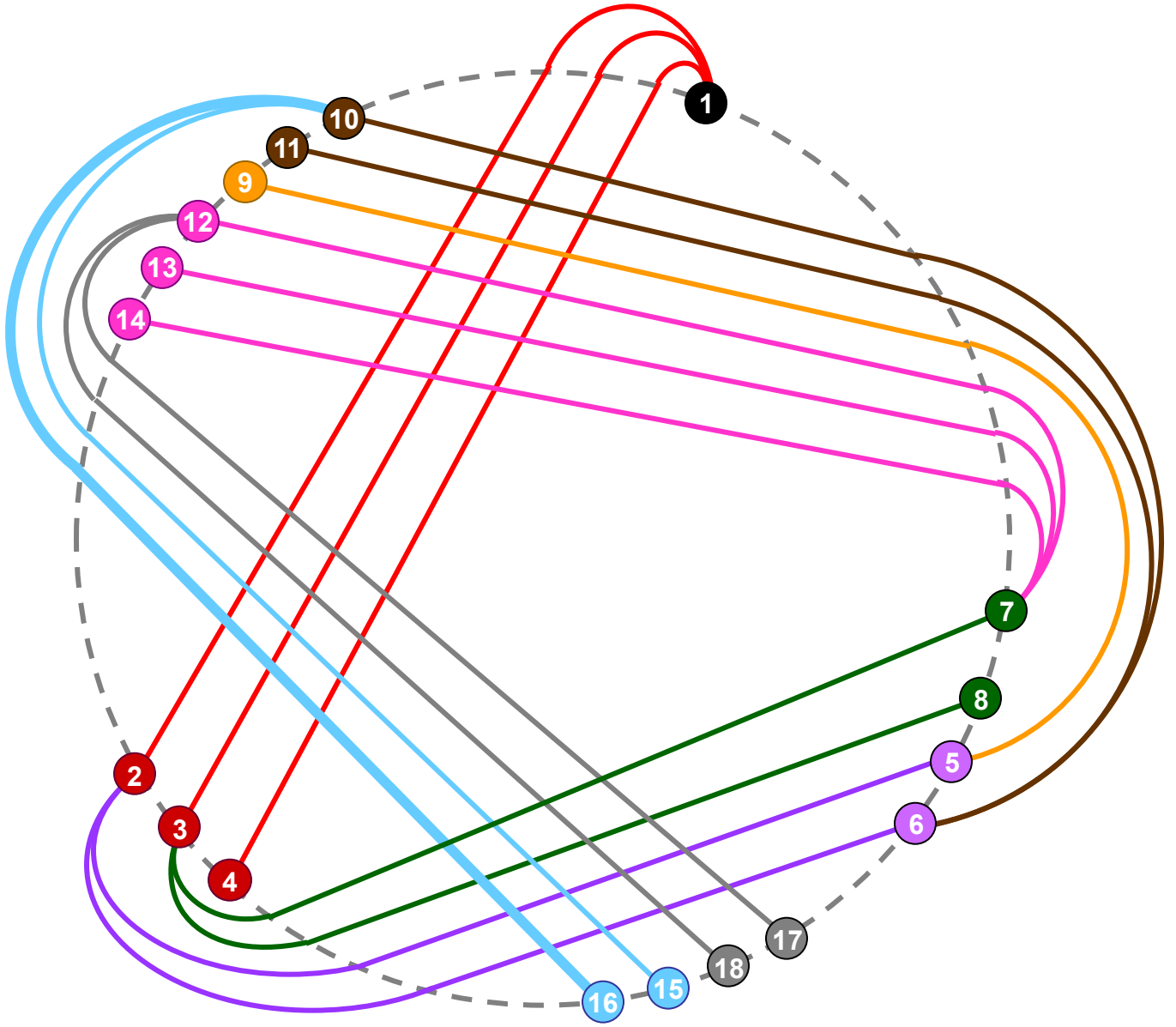
# .....Once every level is a full rainbow....



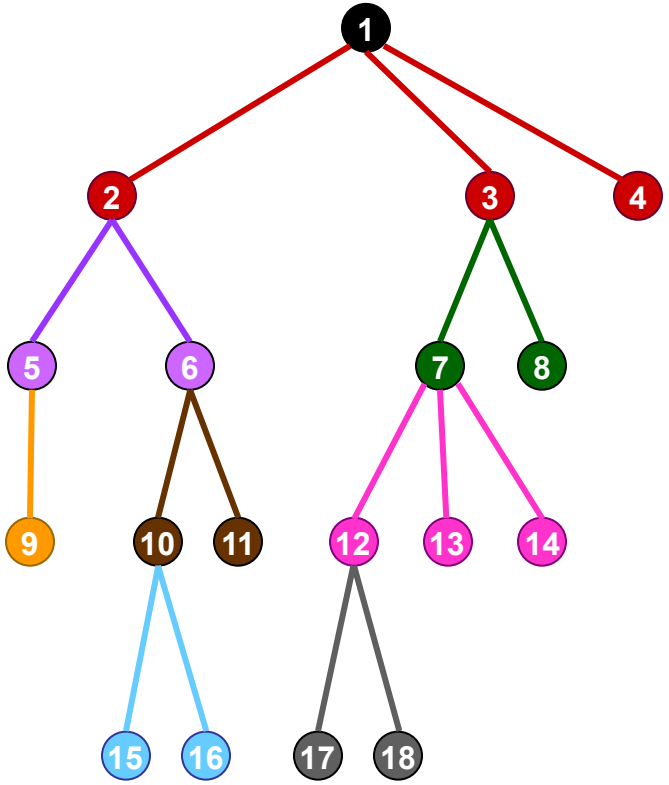



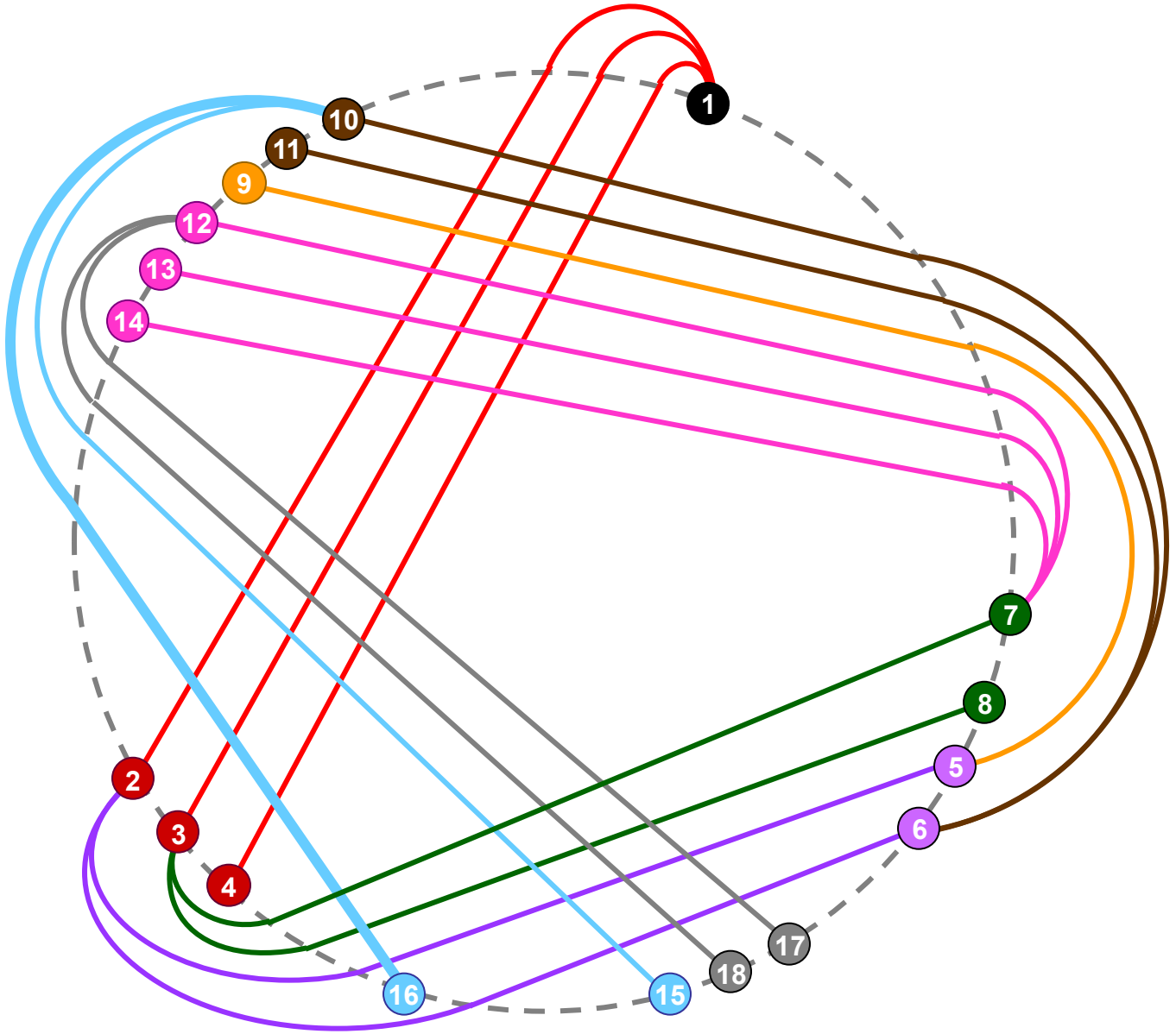
# .....Once every level is a full rainbow....



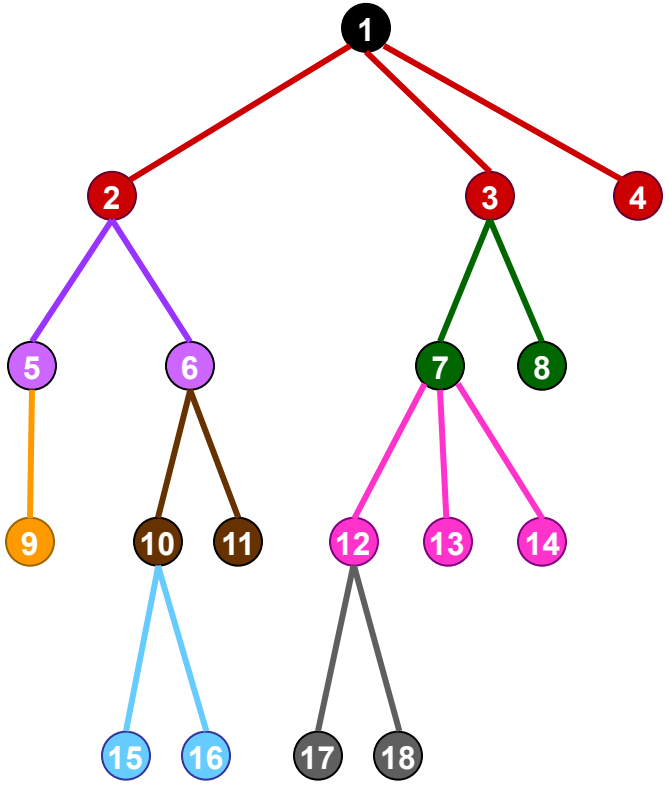



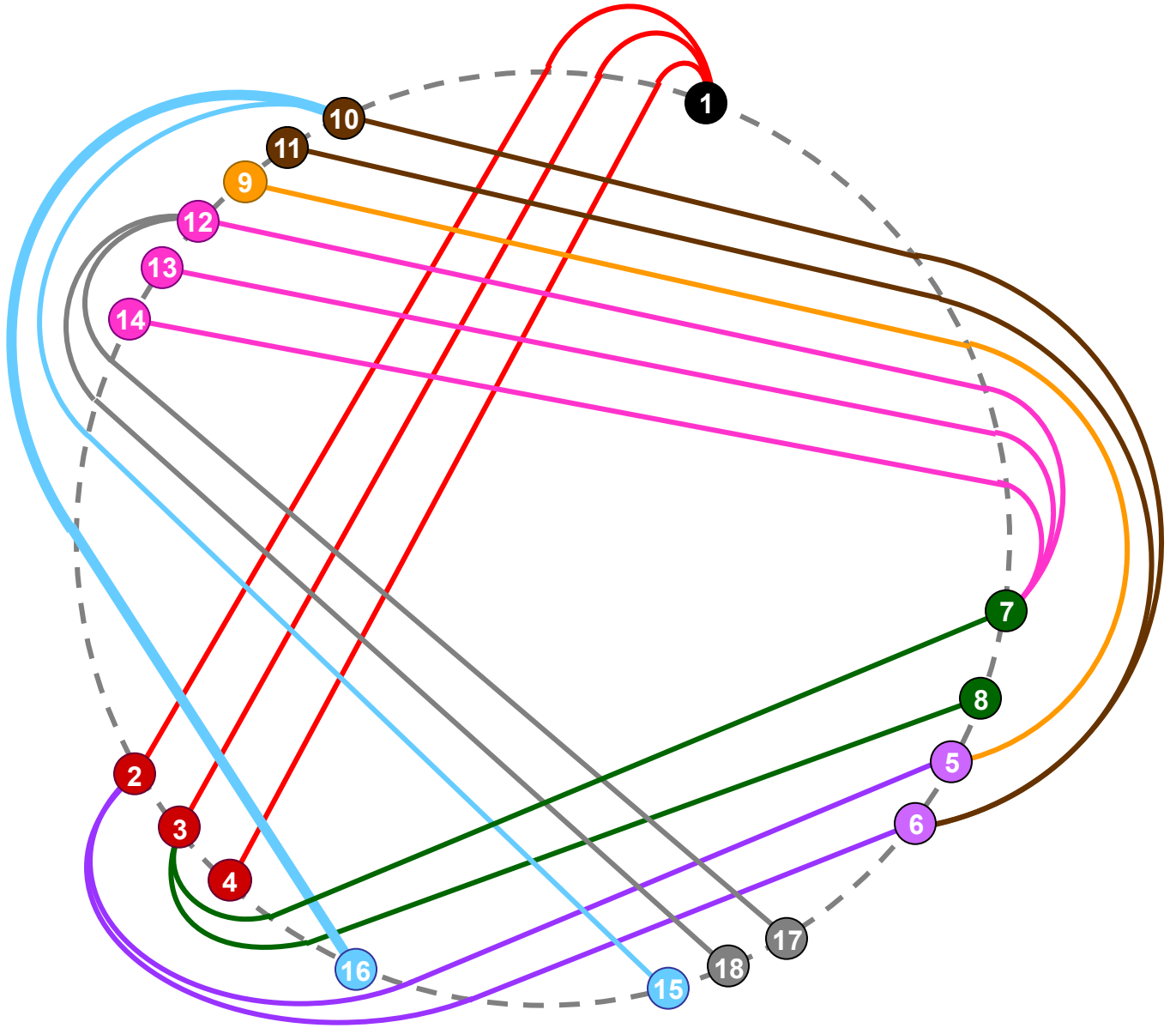
# .....Once every level is a full rainbow....



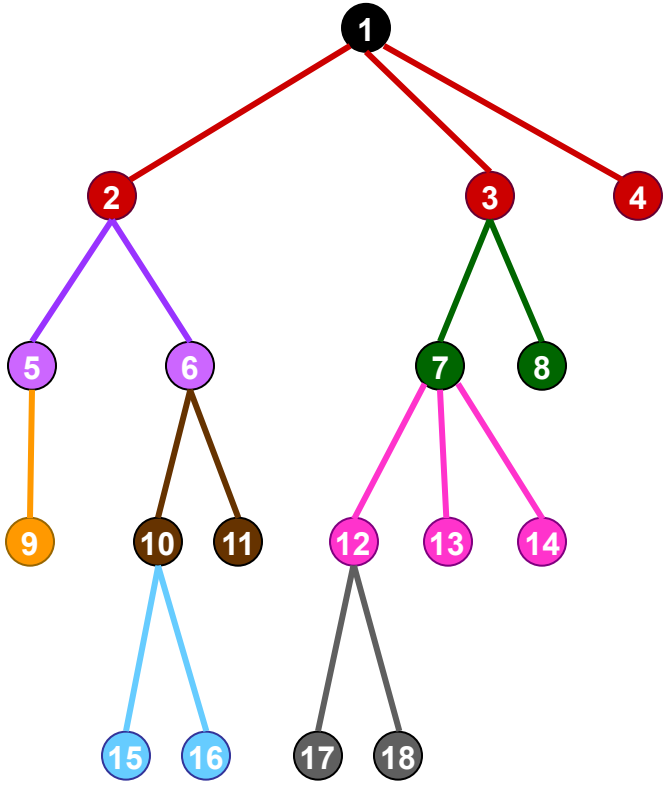



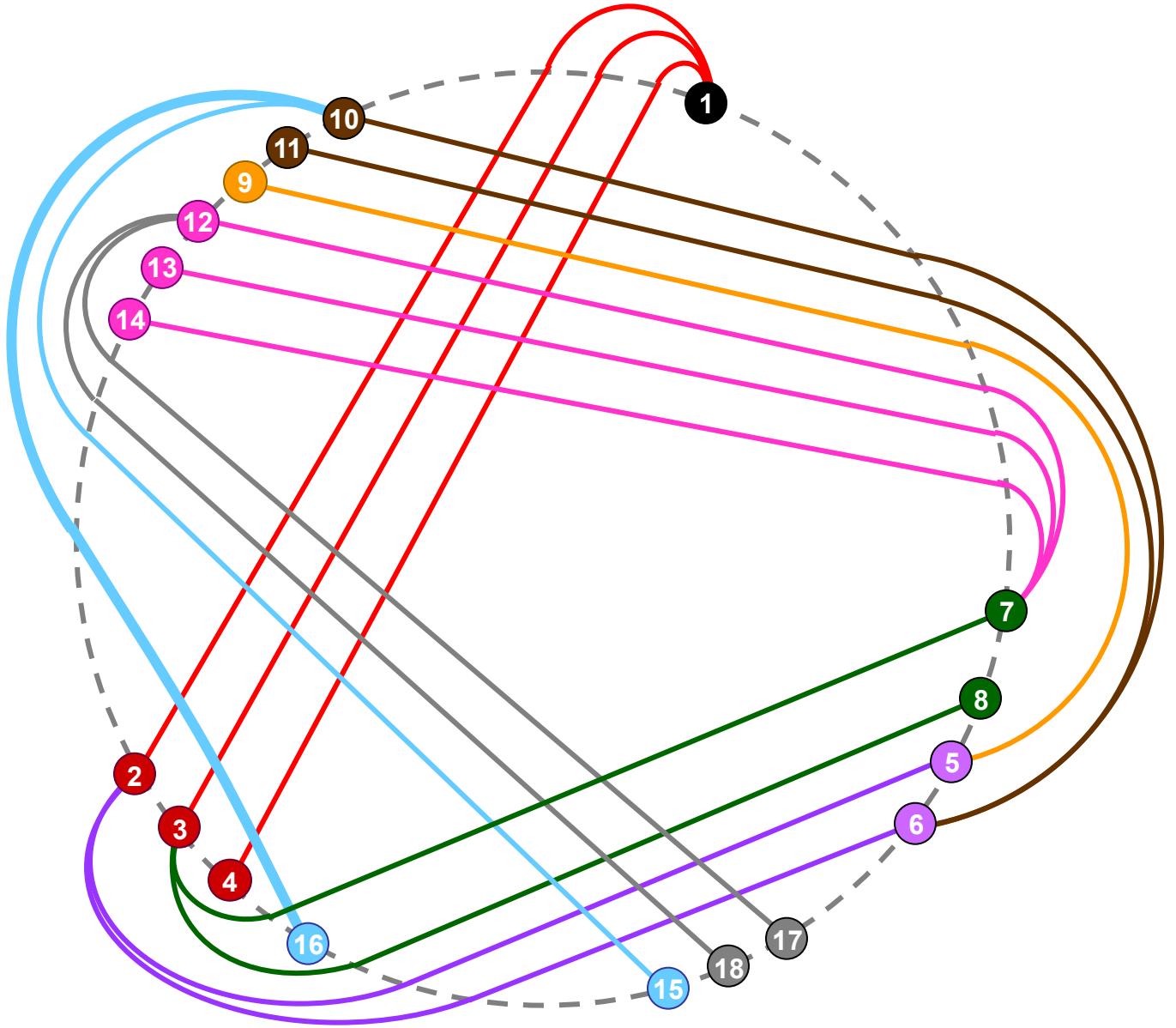
# .....Once every level is a full rainbow....



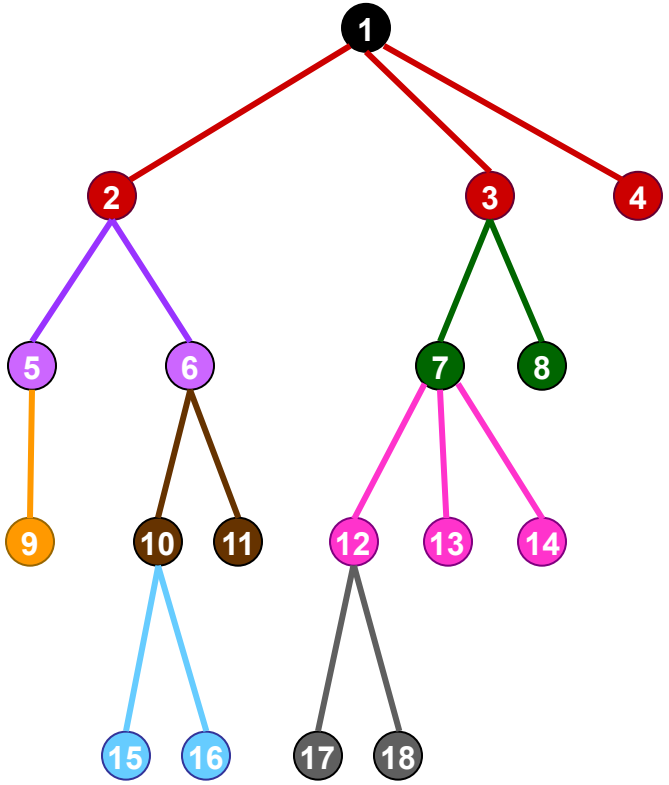



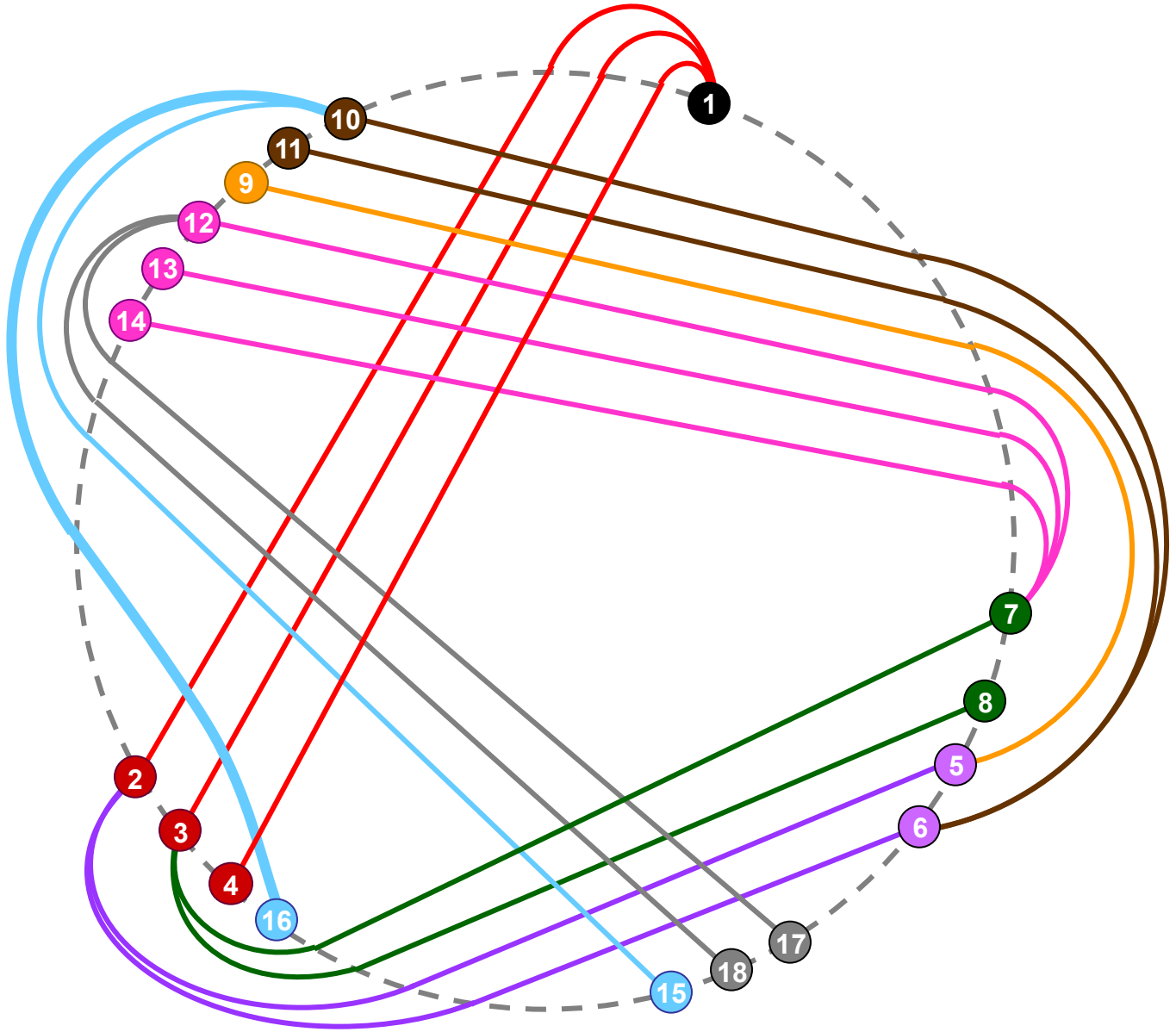
# .....Once every level is a full rainbow....



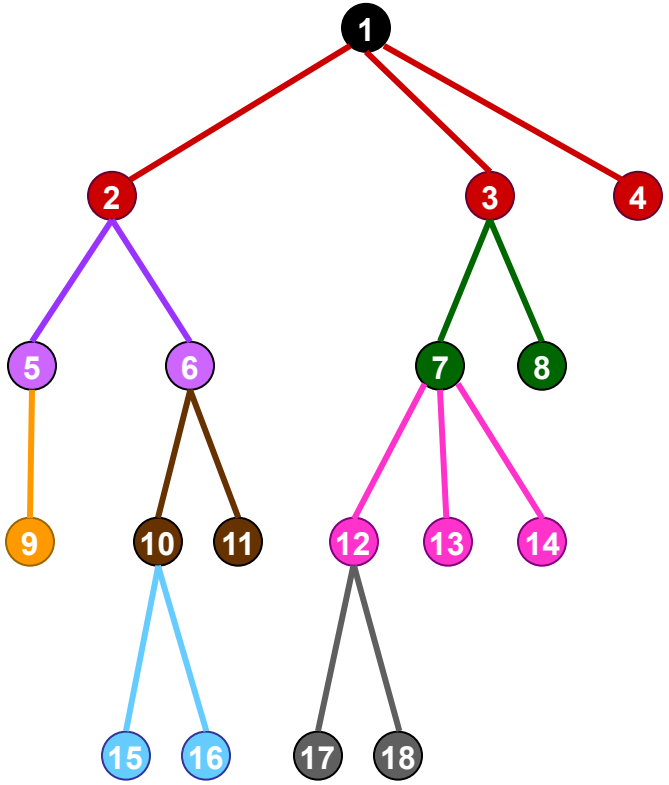



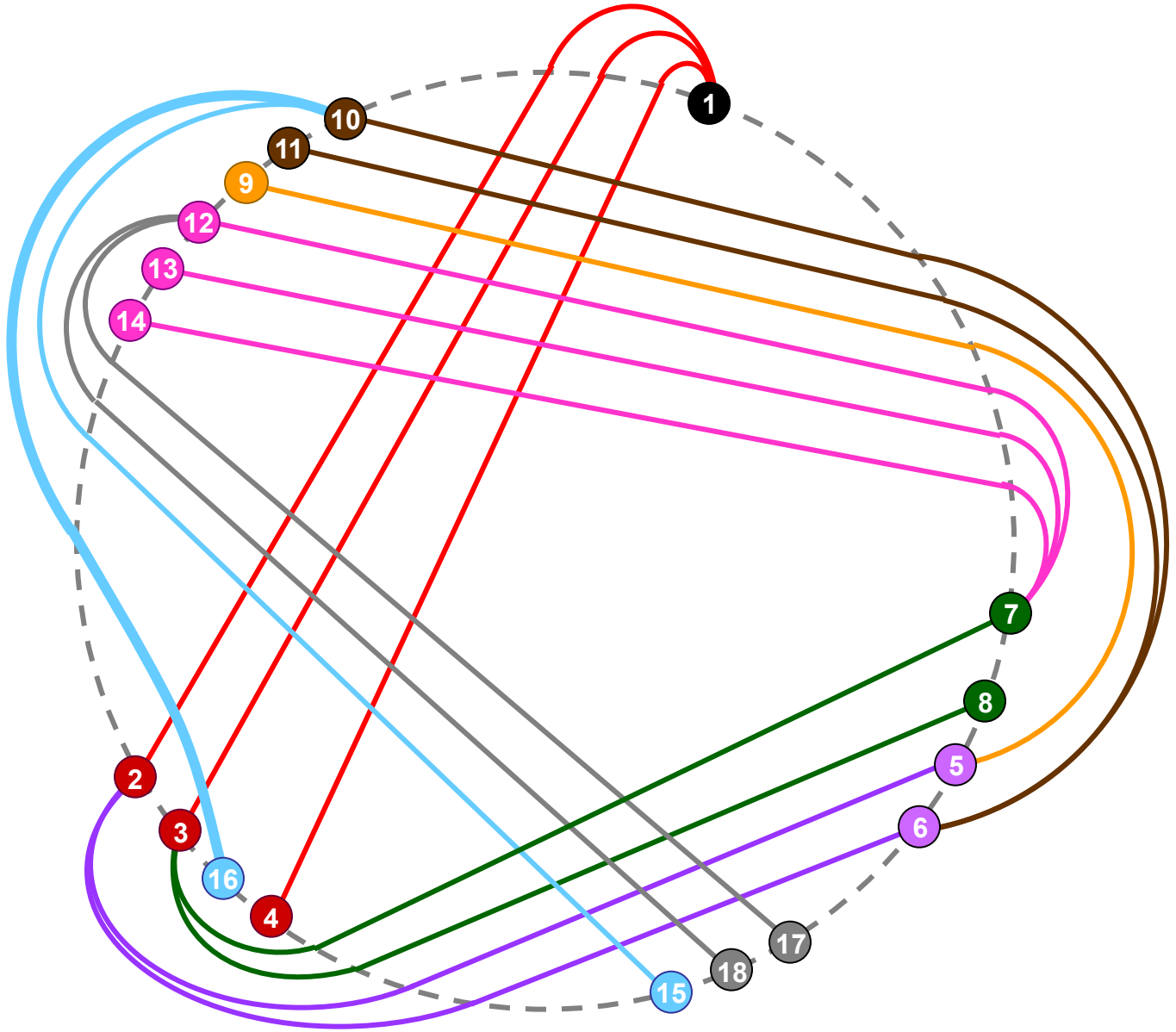
# .....Once every level is a full rainbow....



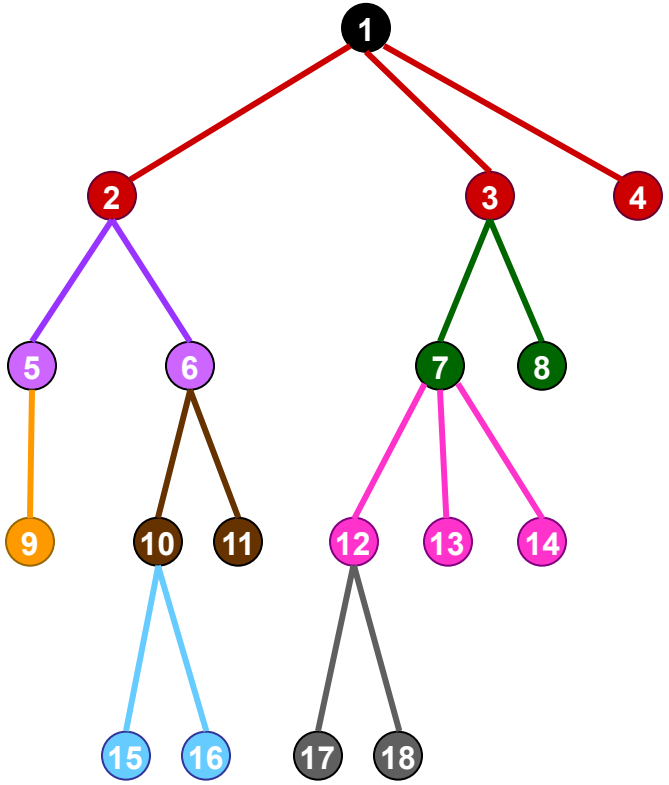



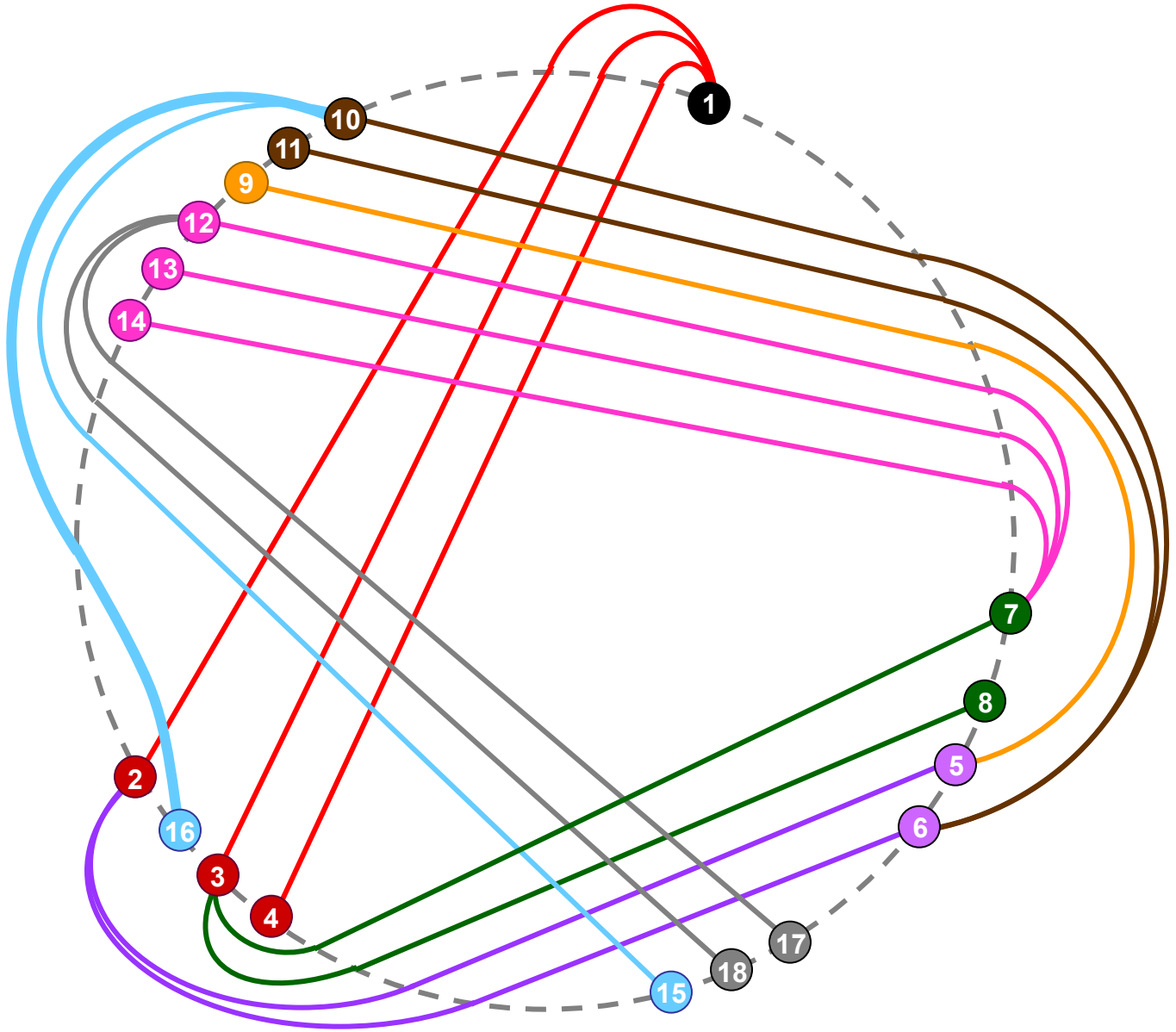
# .....Once every level is a full rainbow....



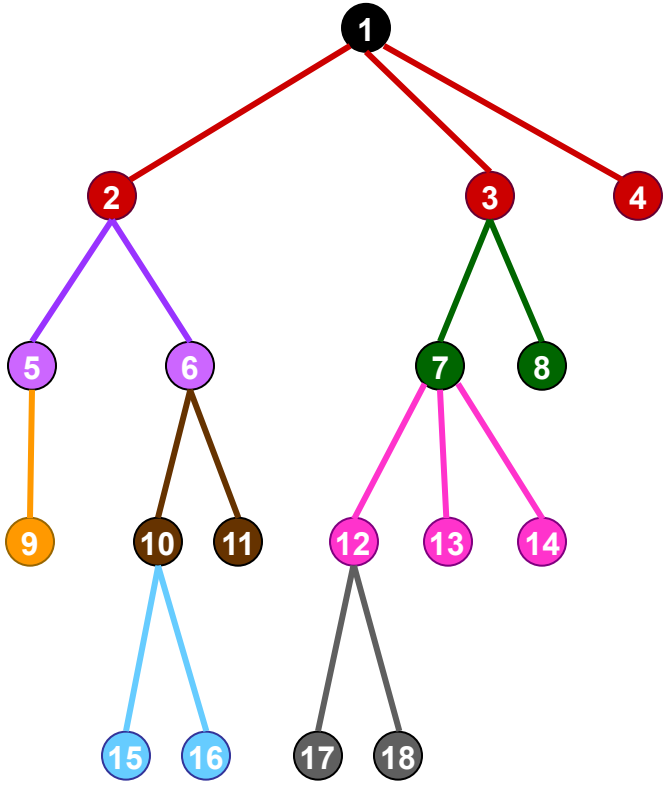



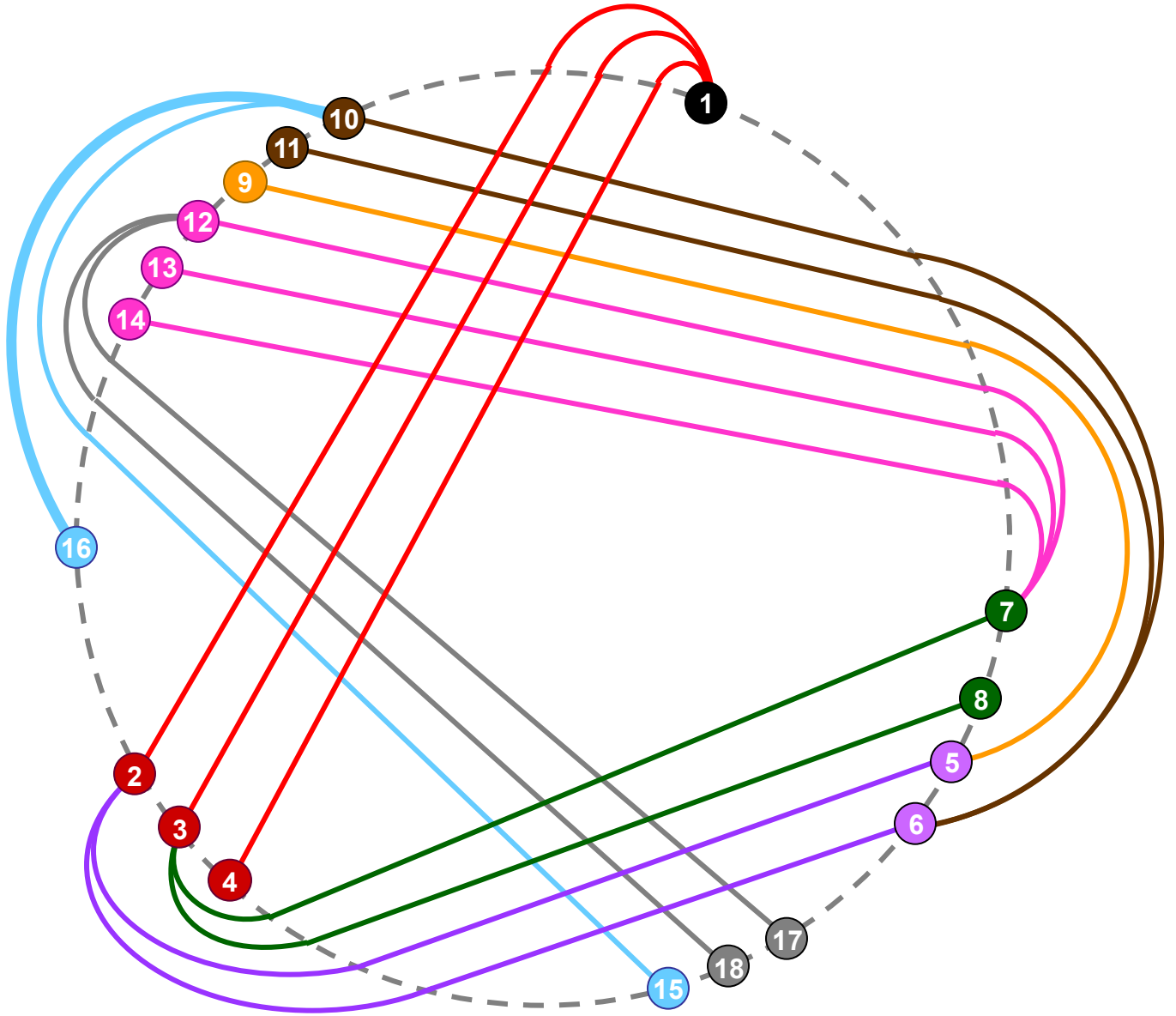
# .....Once every level is a full rainbow....



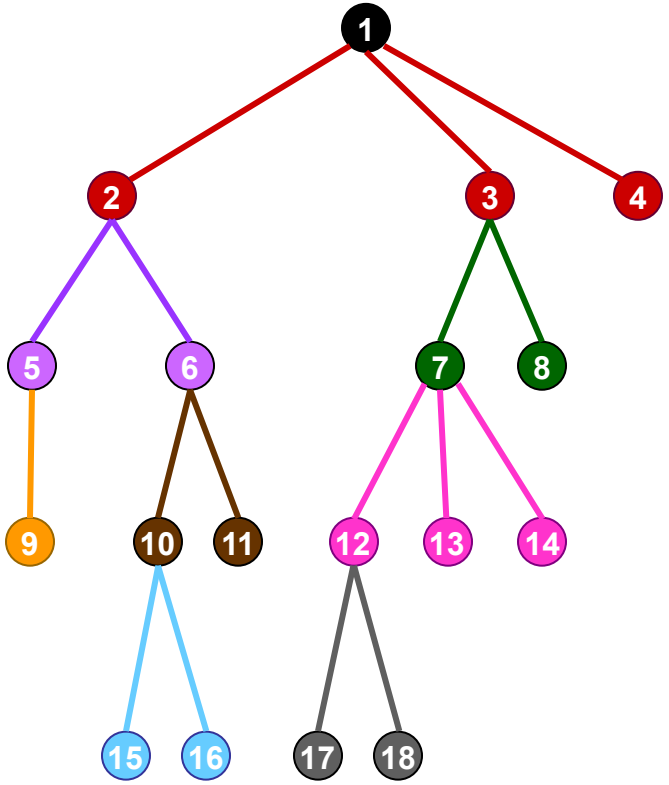



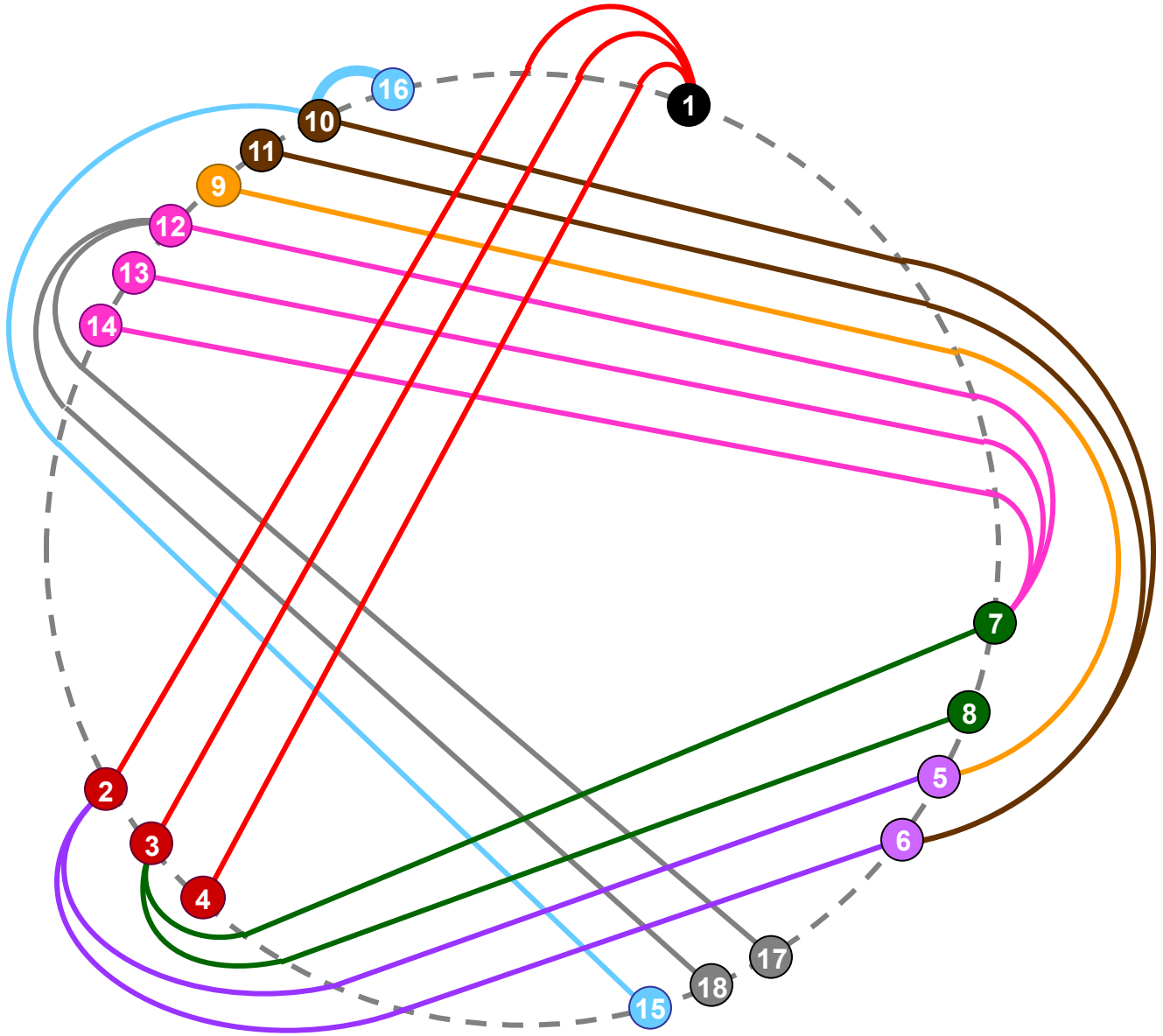
# .....Once every level is a full rainbow....



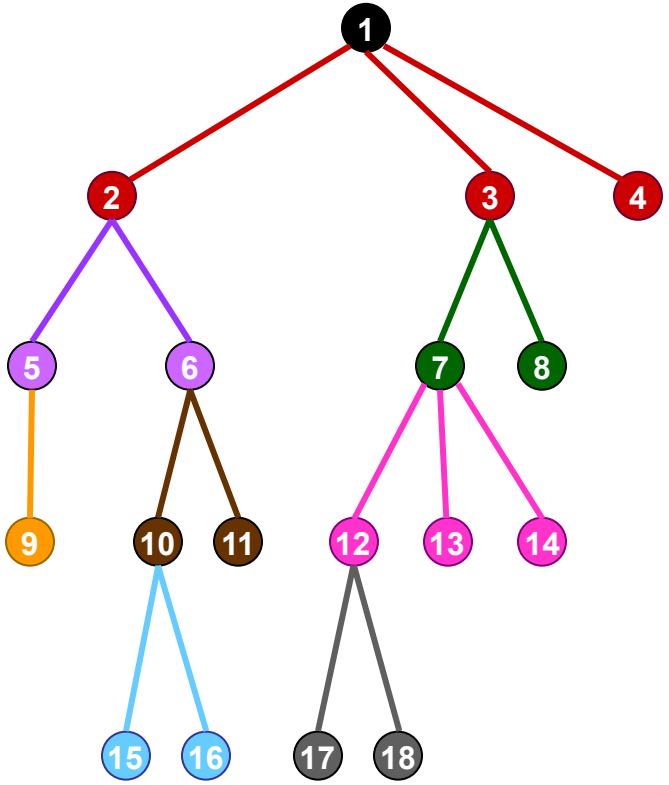



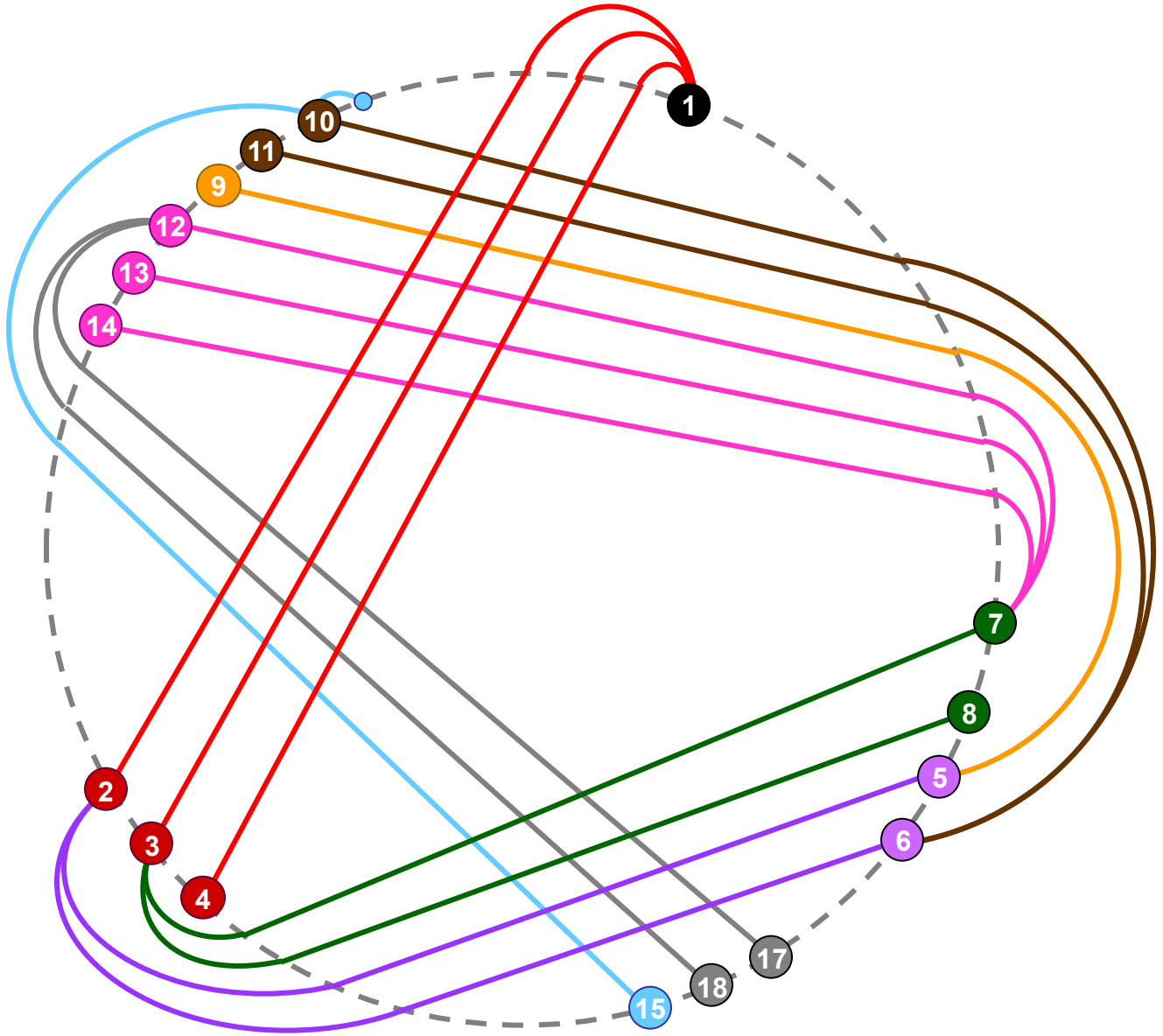
# .....Once every level is a full rainbow....



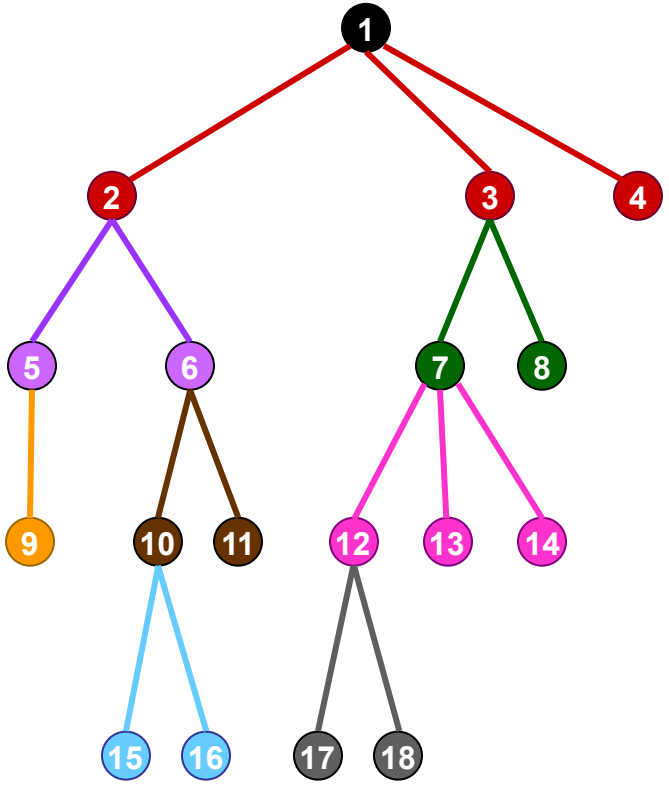



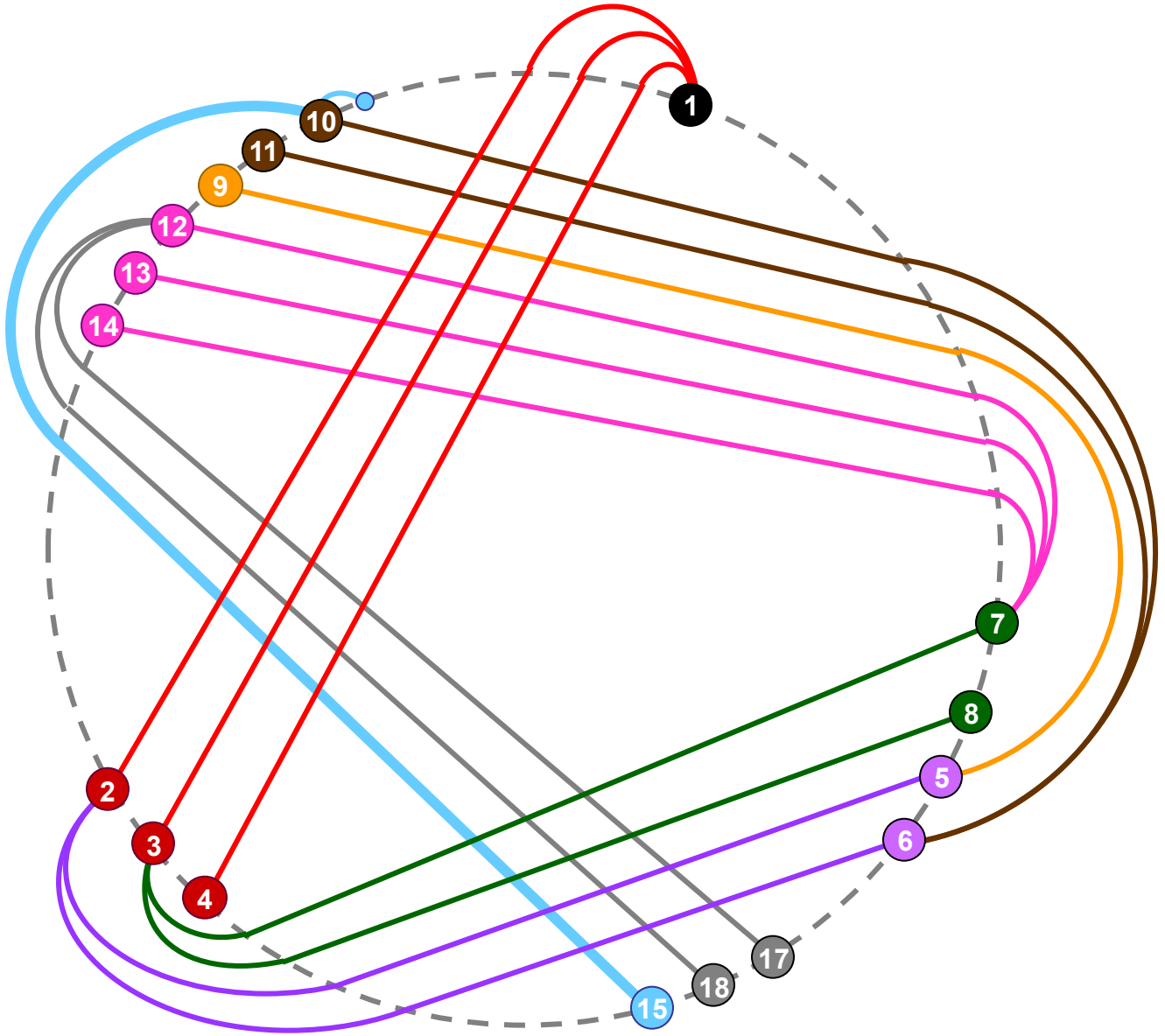
# .....Once every level is a full rainbow....



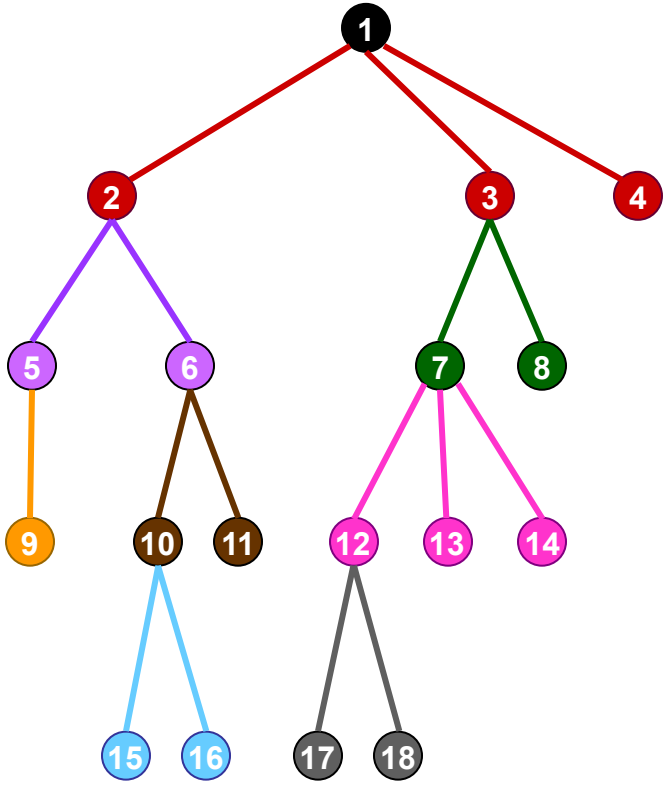



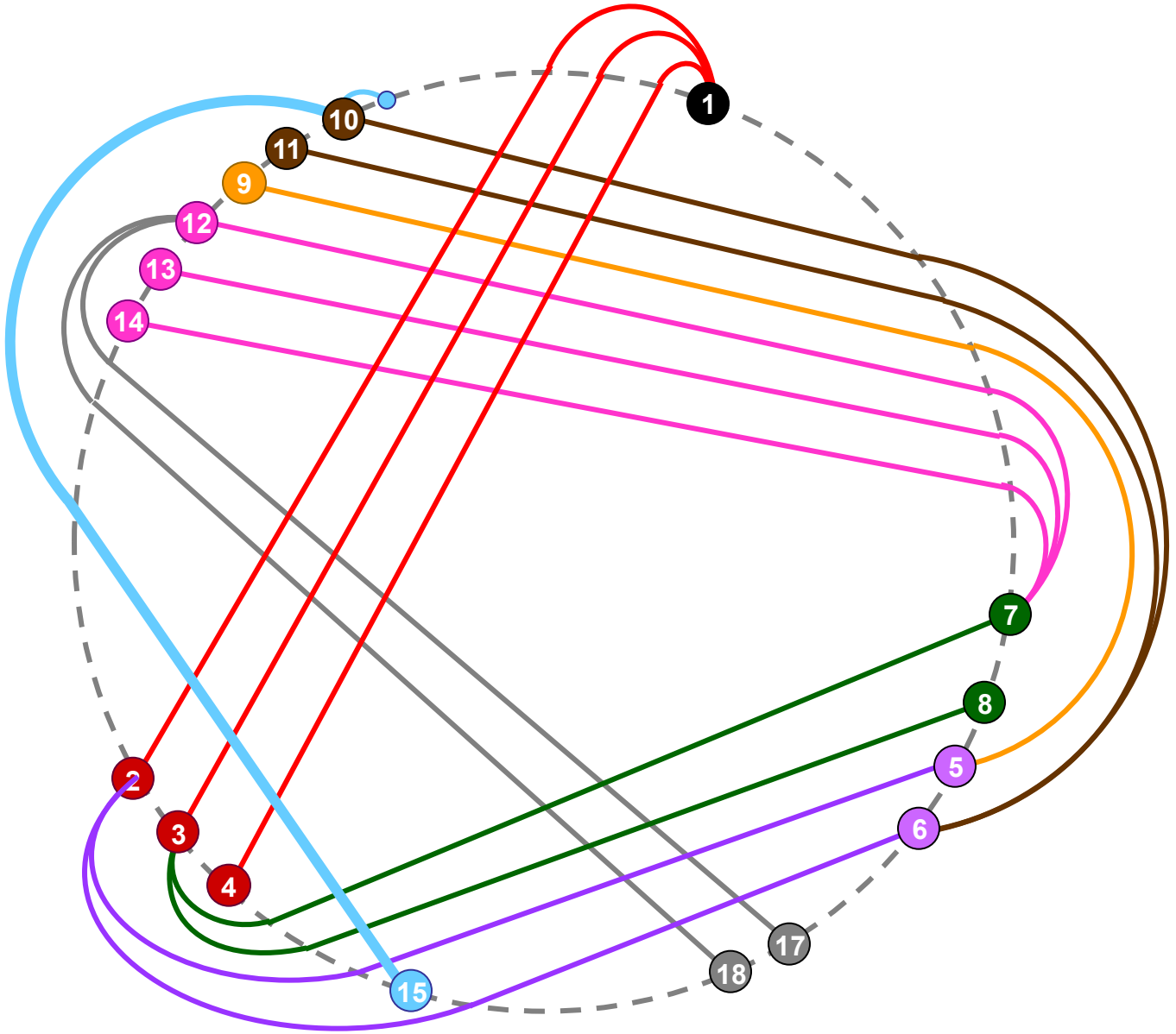
# .....Once every level is a full rainbow....



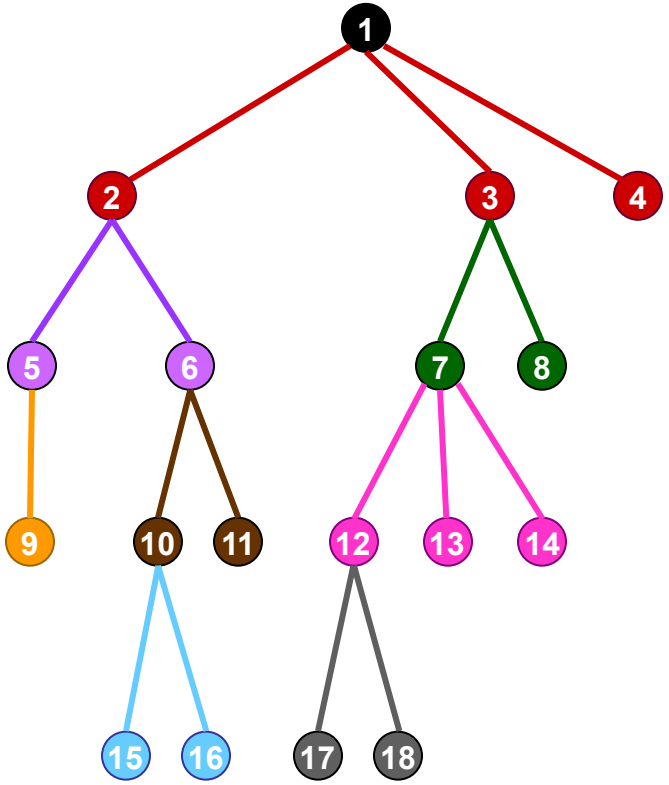



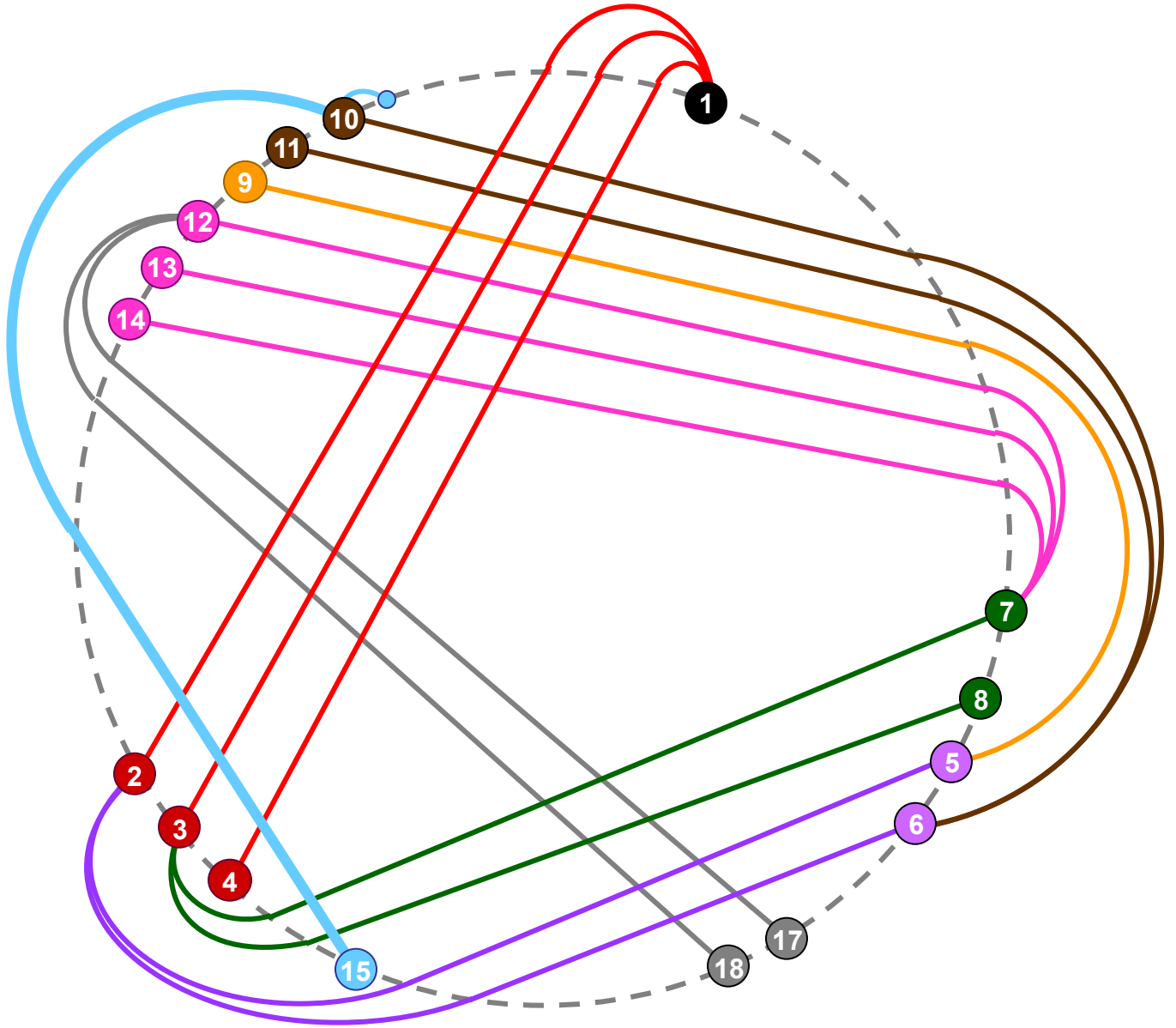
# .....Once every level is a full rainbow....



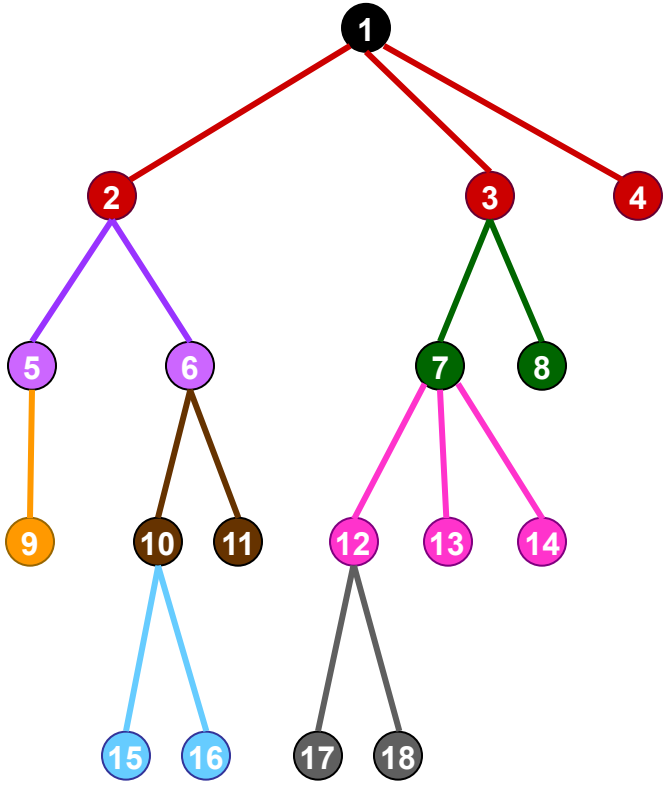



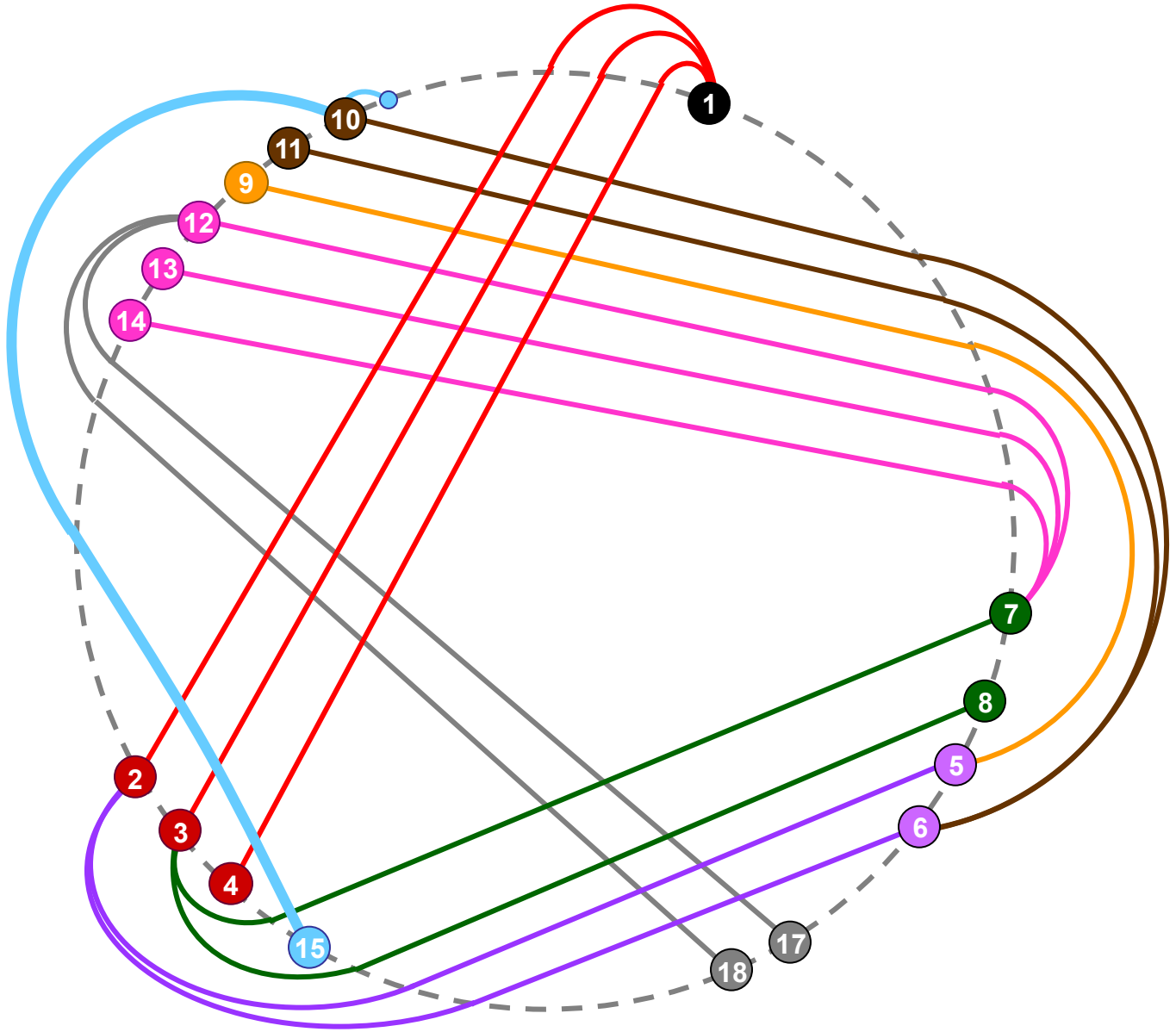
# .....Once every level is a full rainbow....



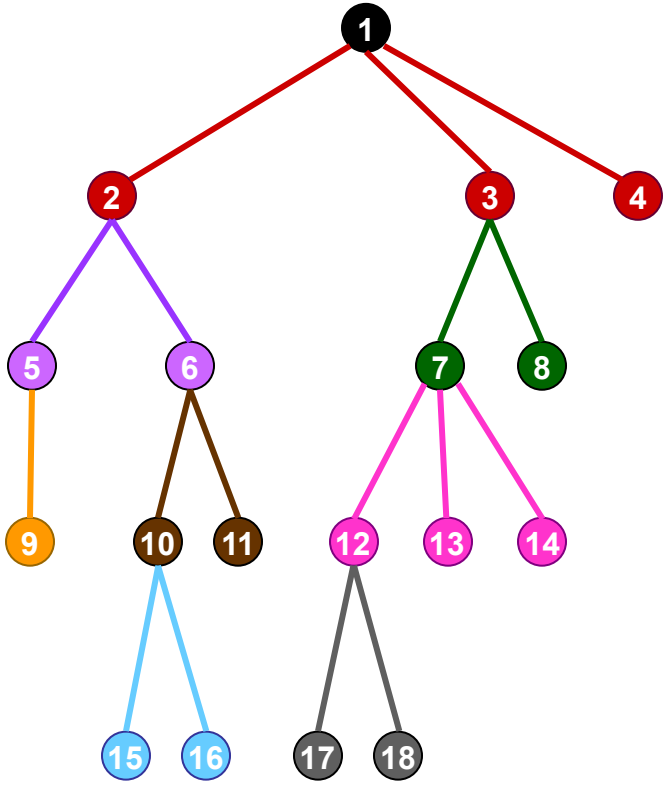



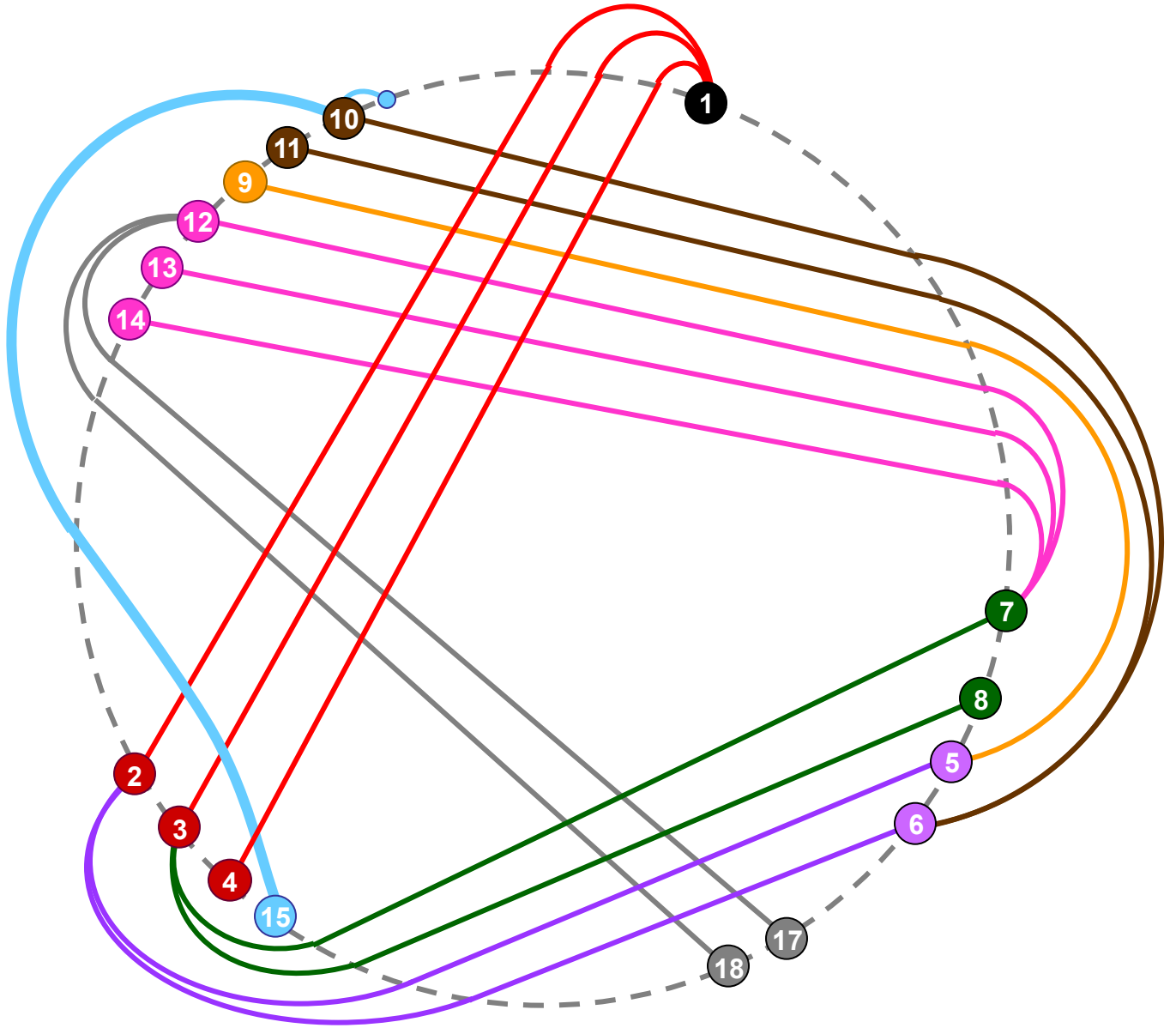
# .....Once every level is a full rainbow....



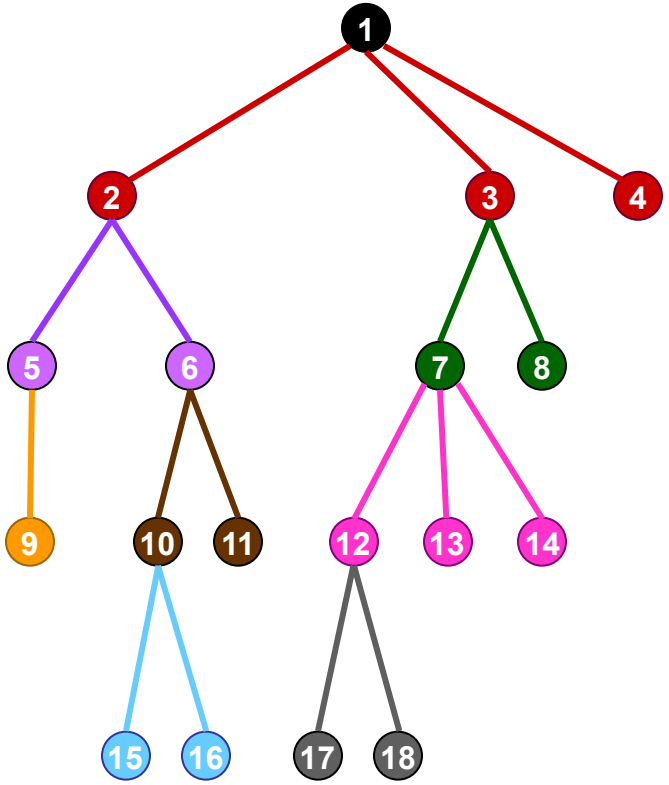



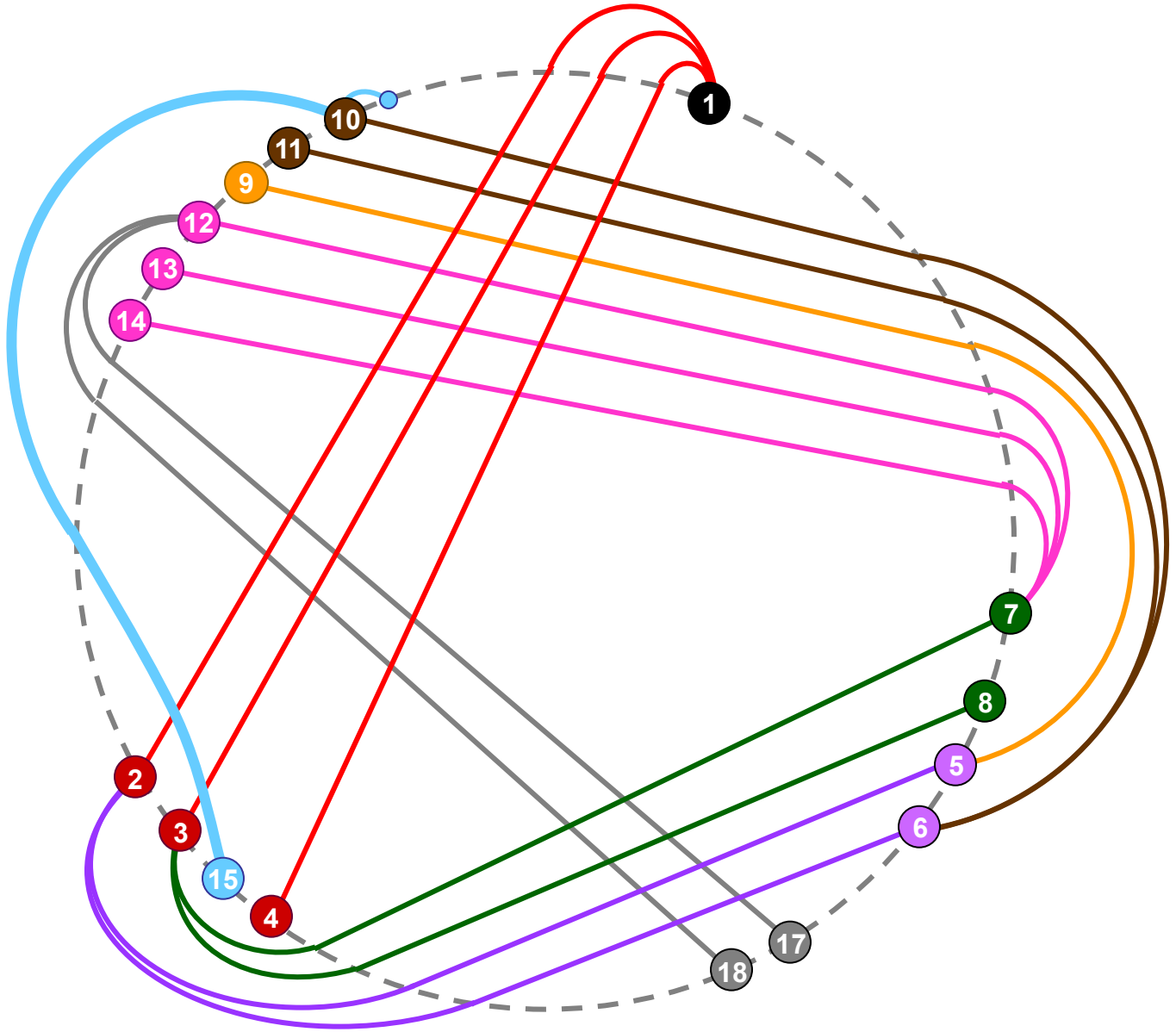
# .....Once every level is a full rainbow....



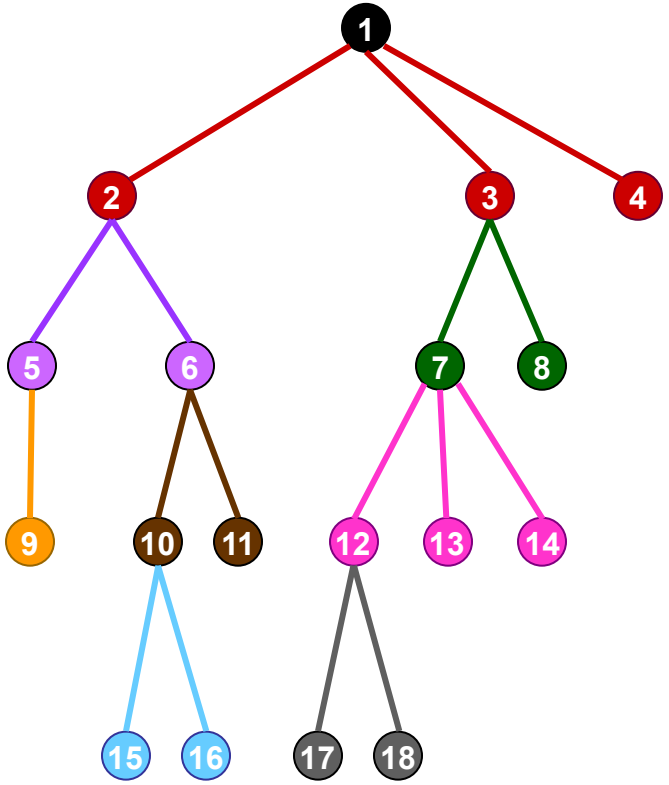



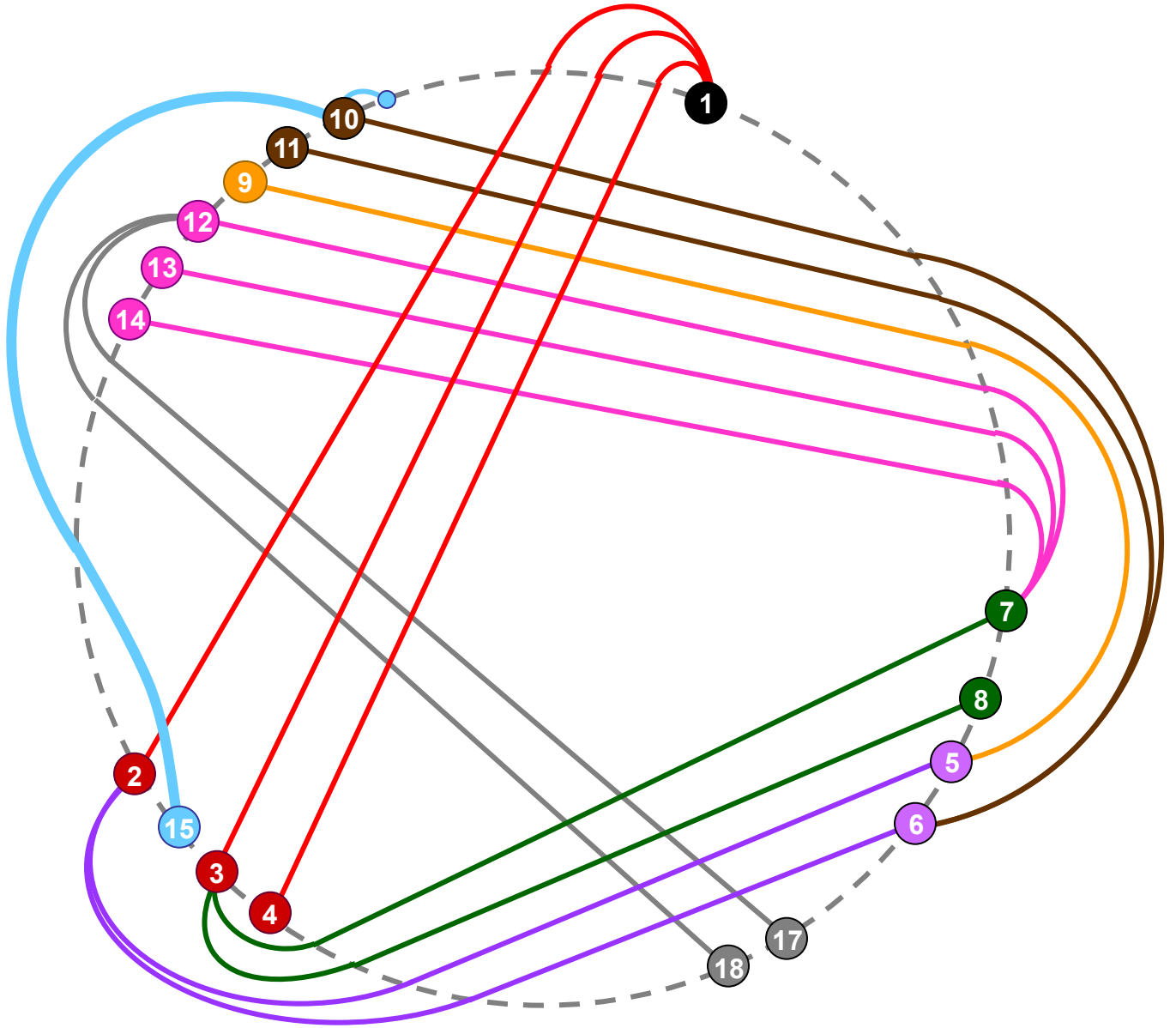
# .....Once every level is a full rainbow....



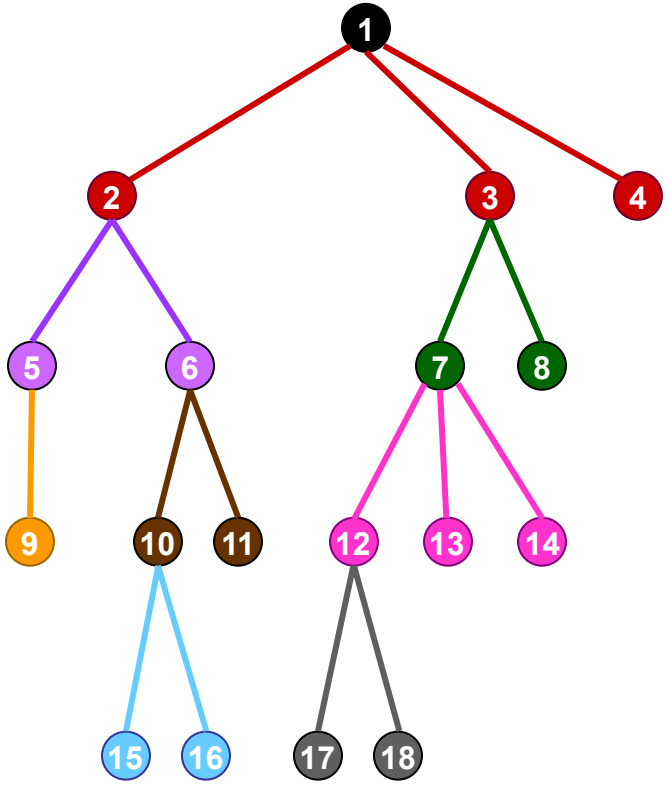



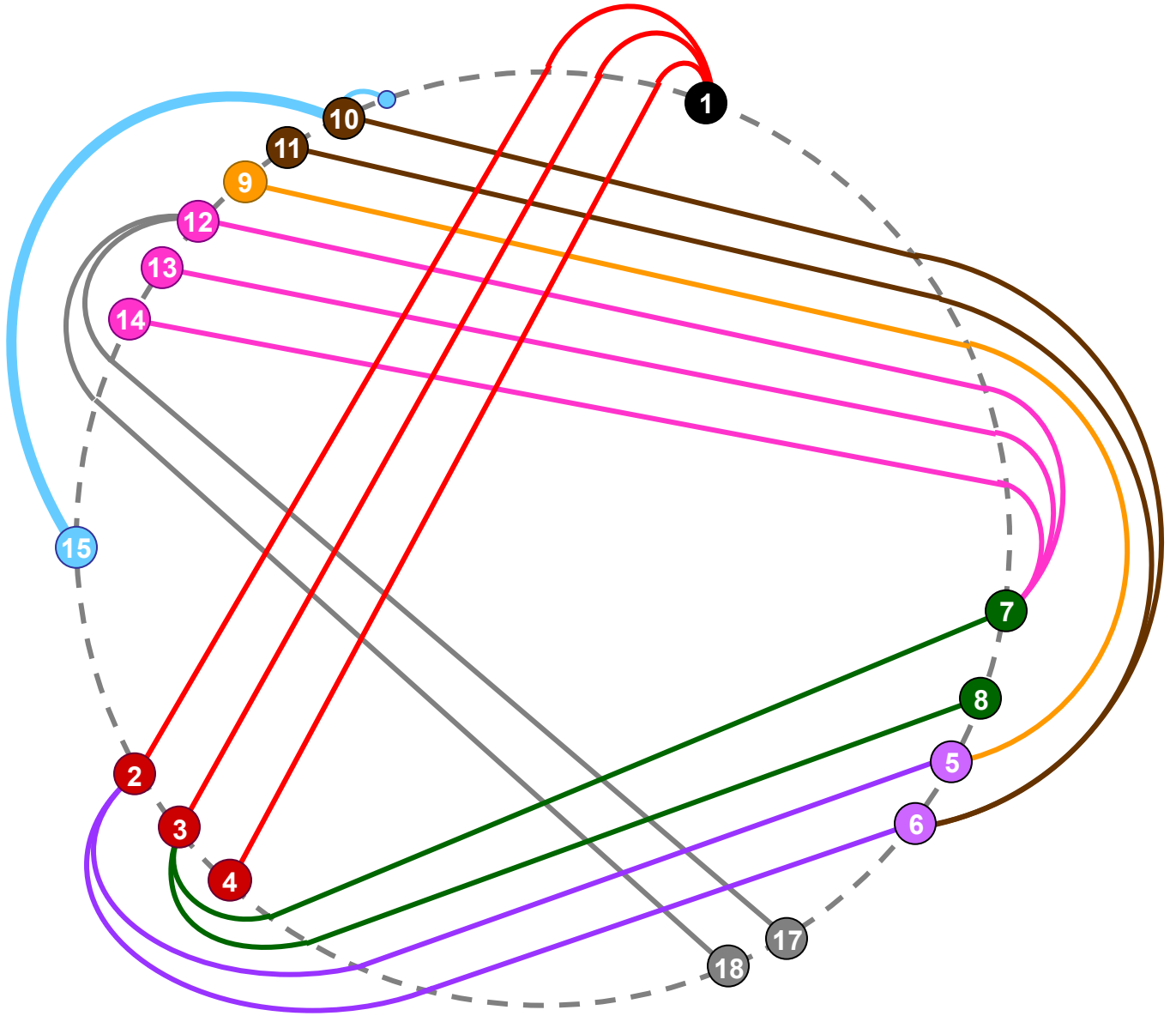
# .....Once every level is a full rainbow....



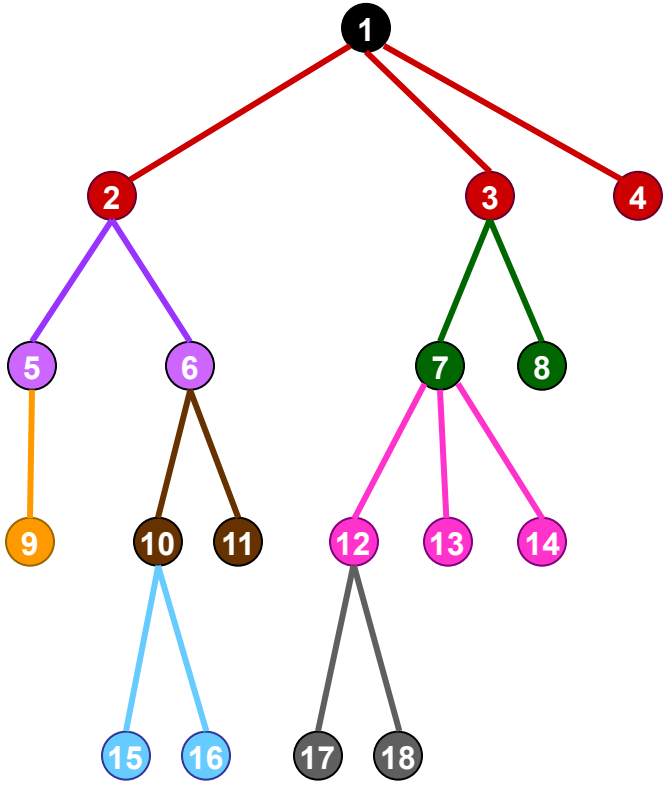



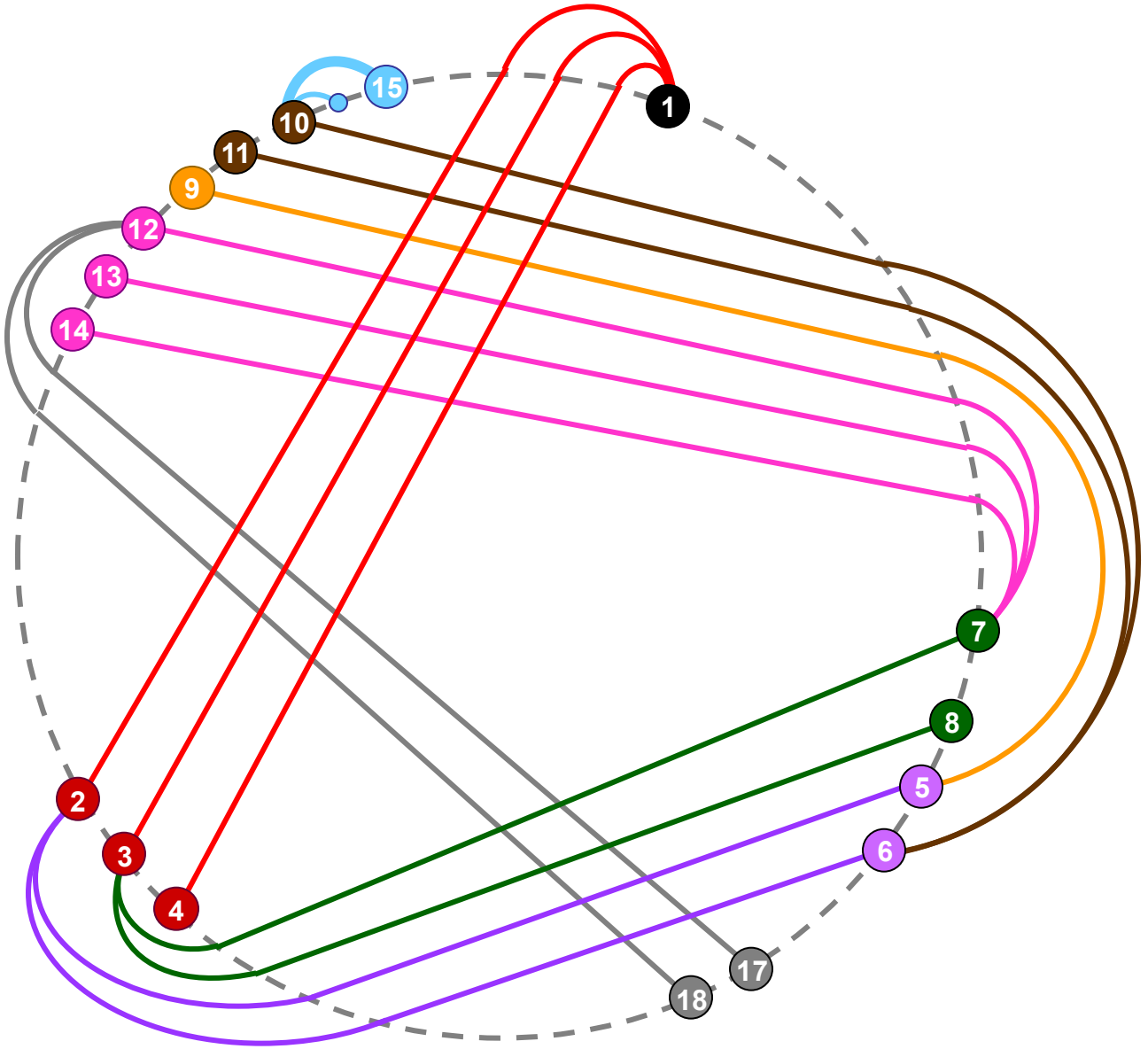
# .....Once every level is a full rainbow....



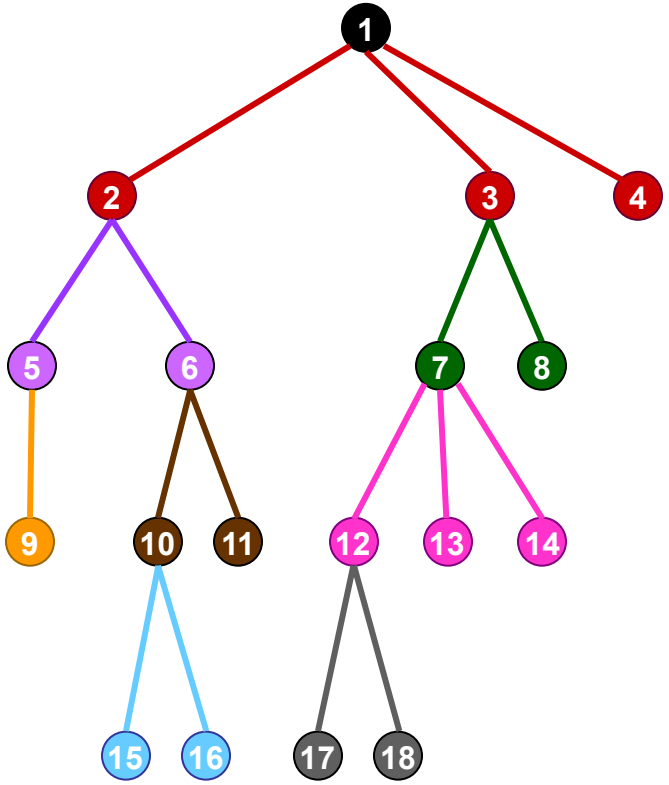



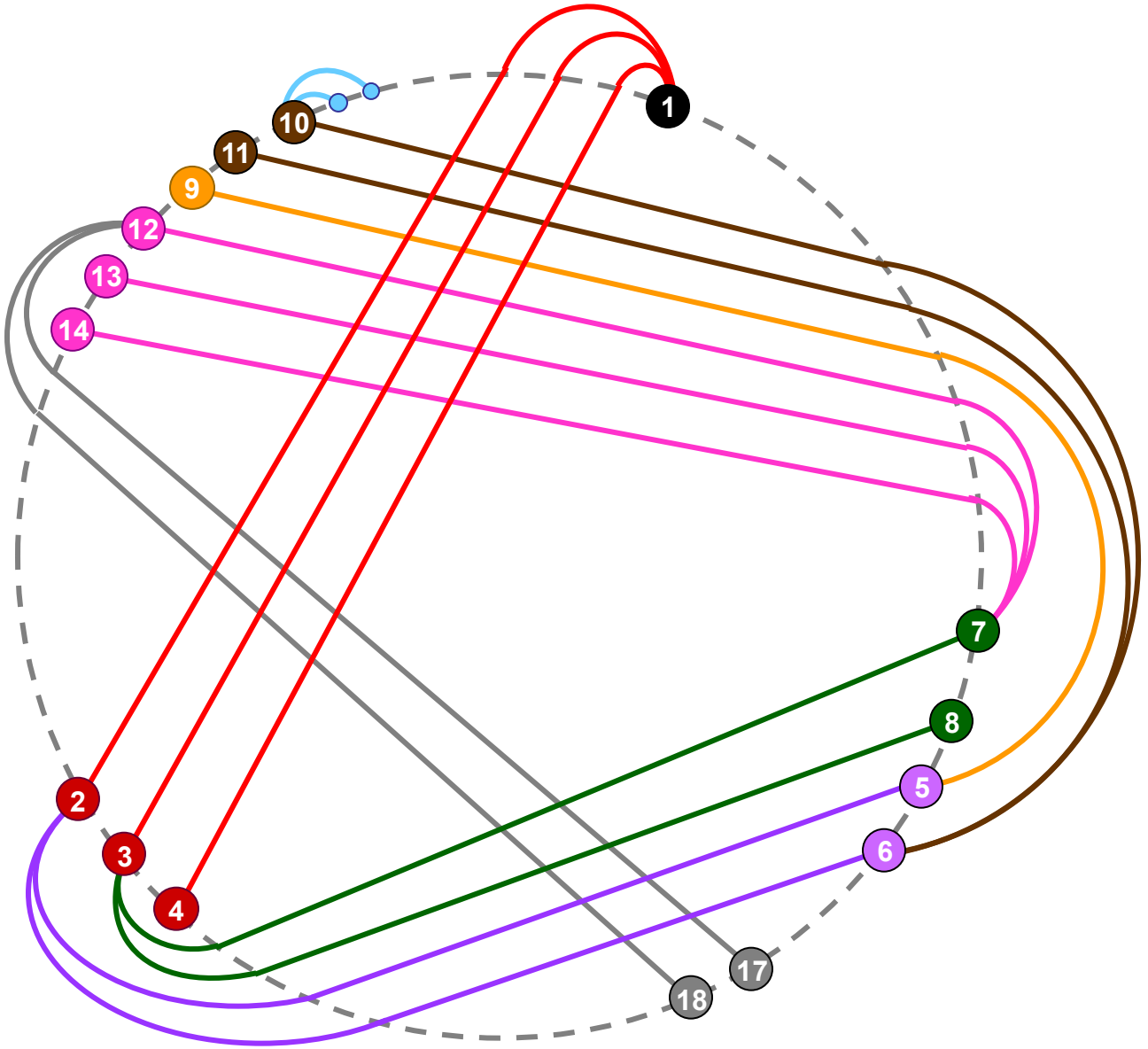
# .....Once every level is a full rainbow....



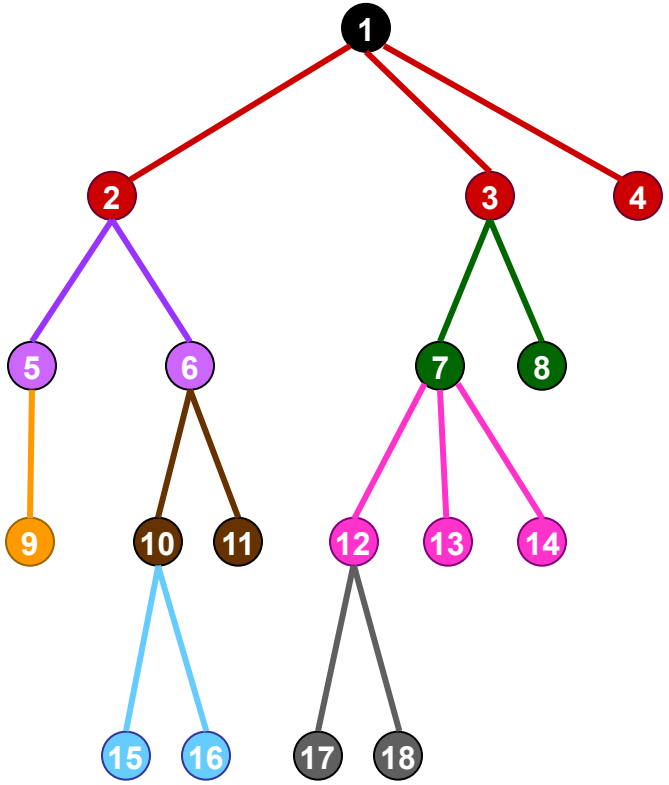



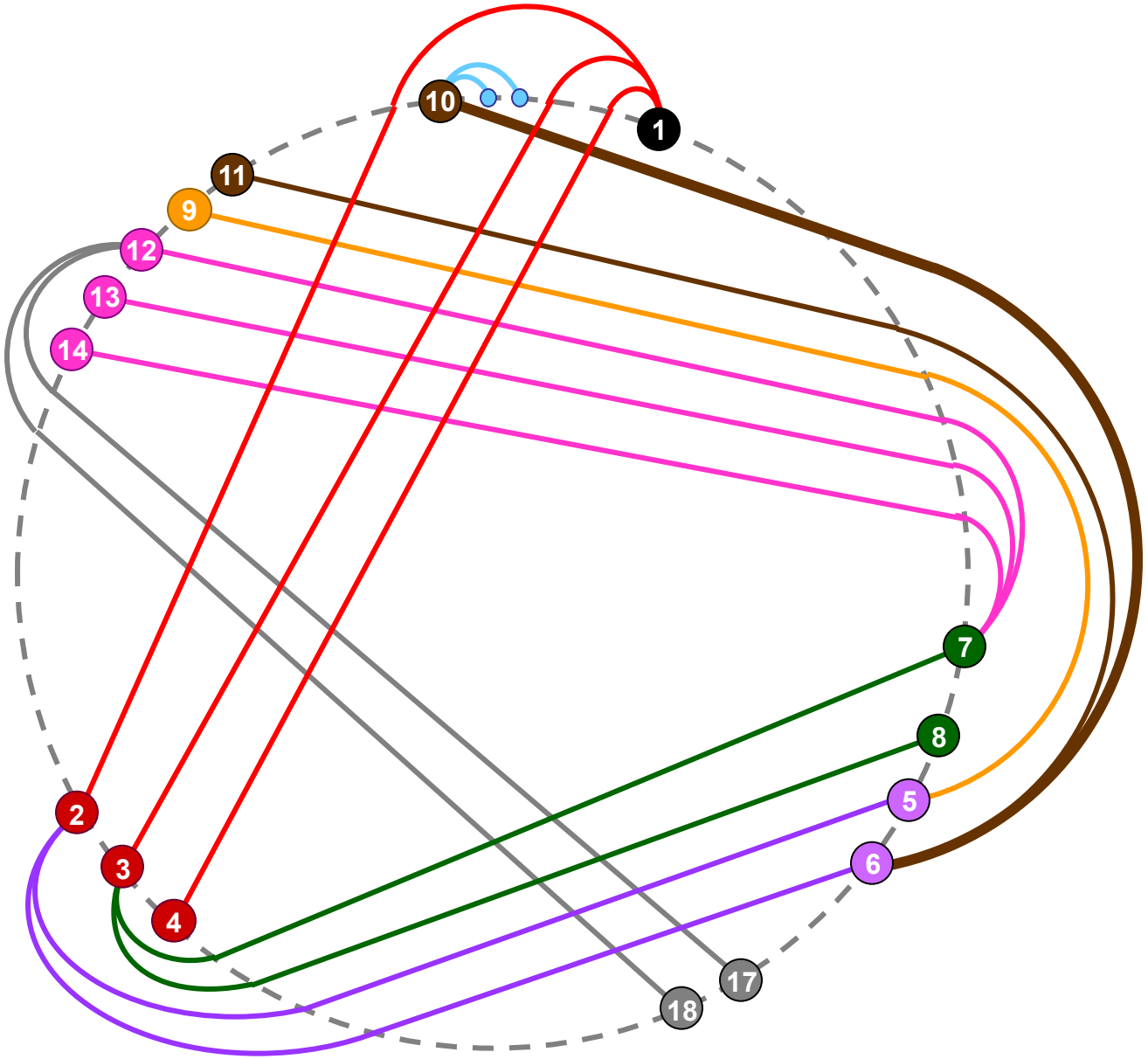
# .....Once every level is a full rainbow....



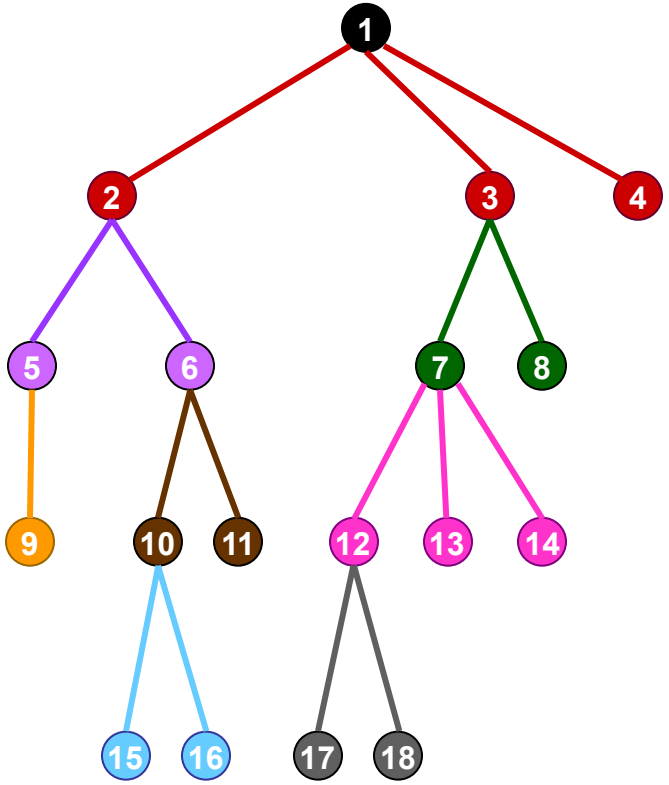



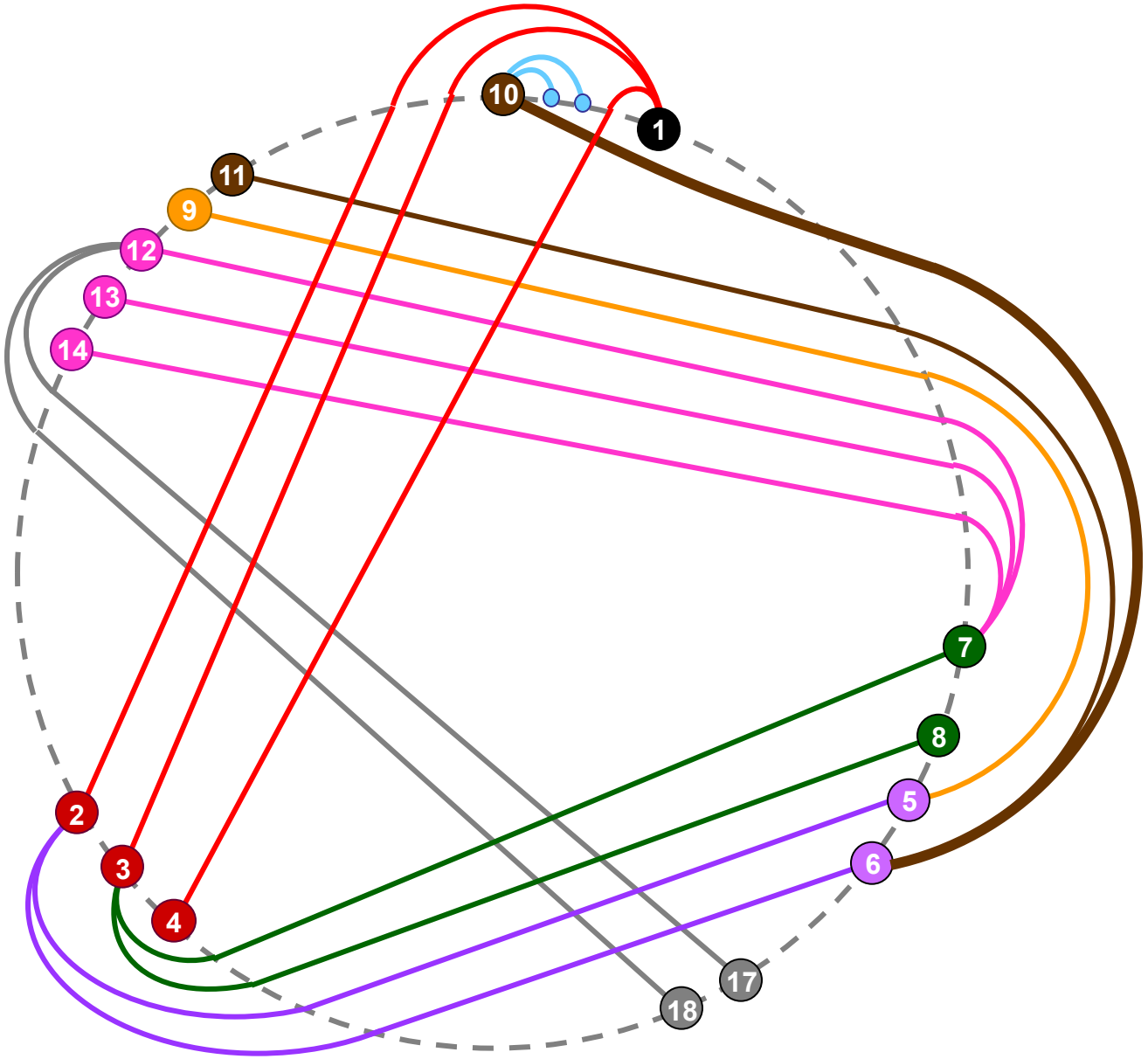
# .....Once every level is a full rainbow....



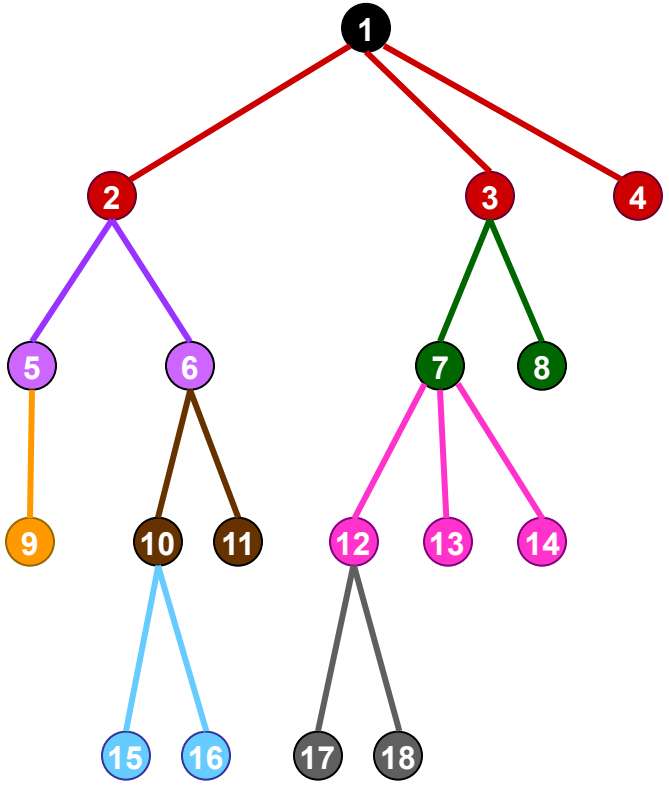



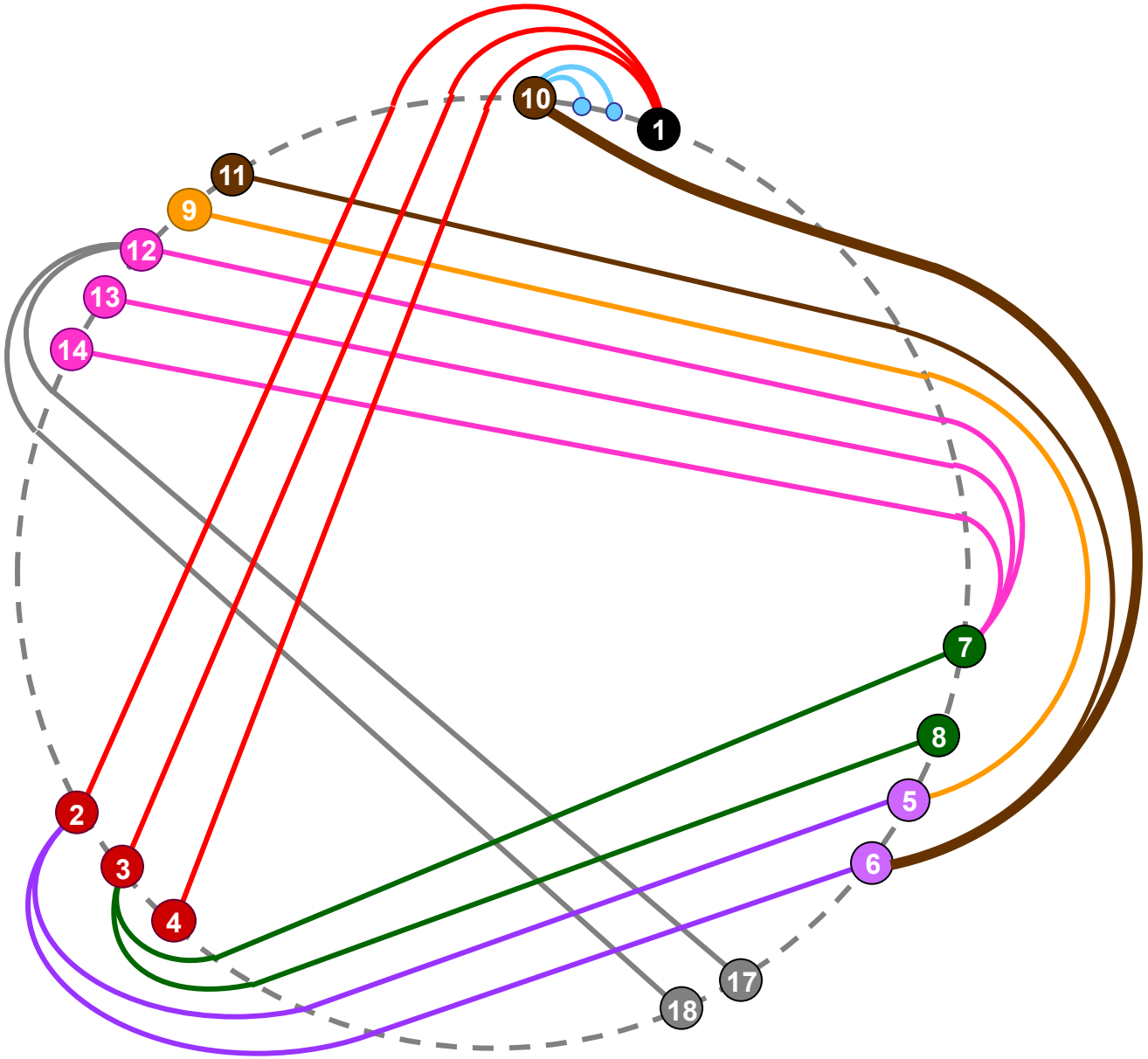
# .....Once every level is a full rainbow....



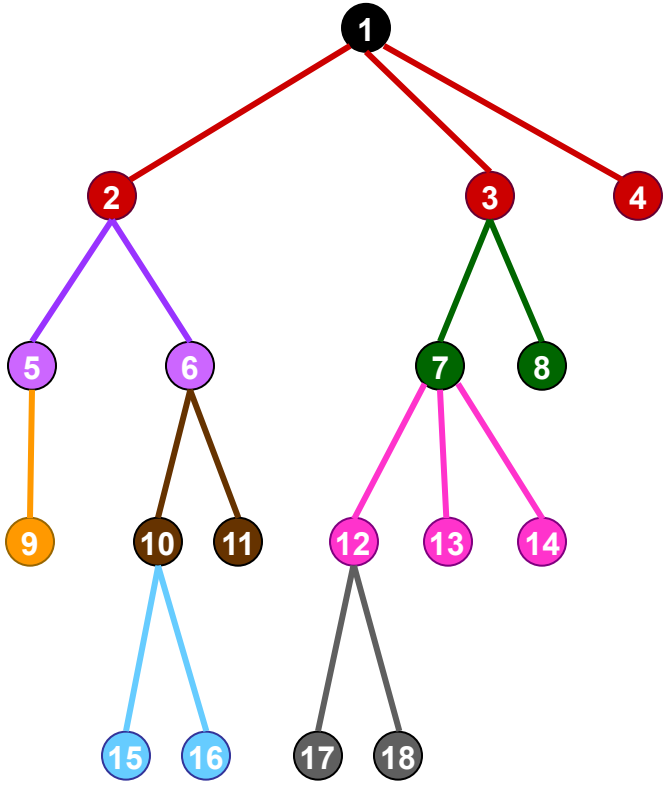



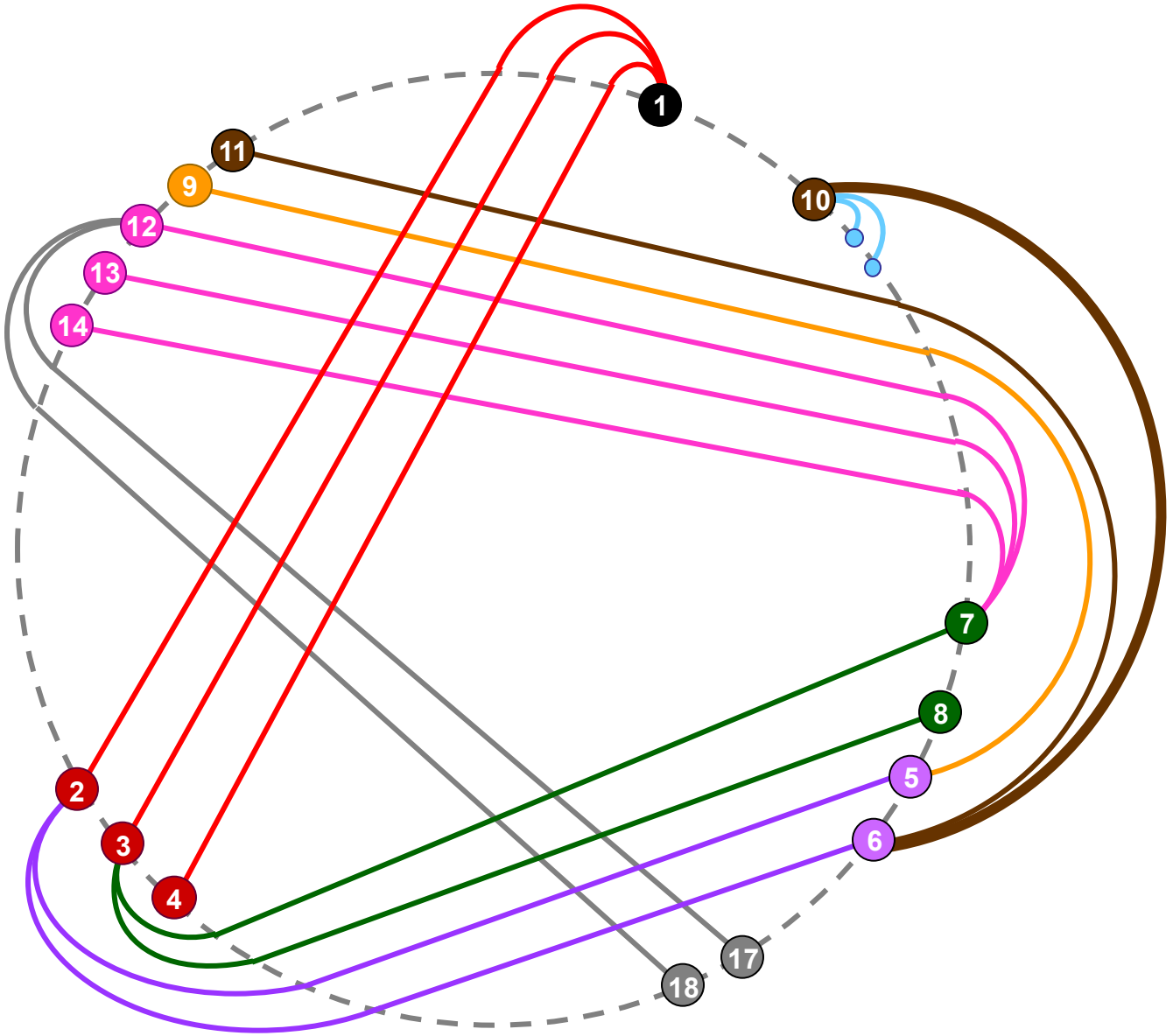
# .....Once every level is a full rainbow....



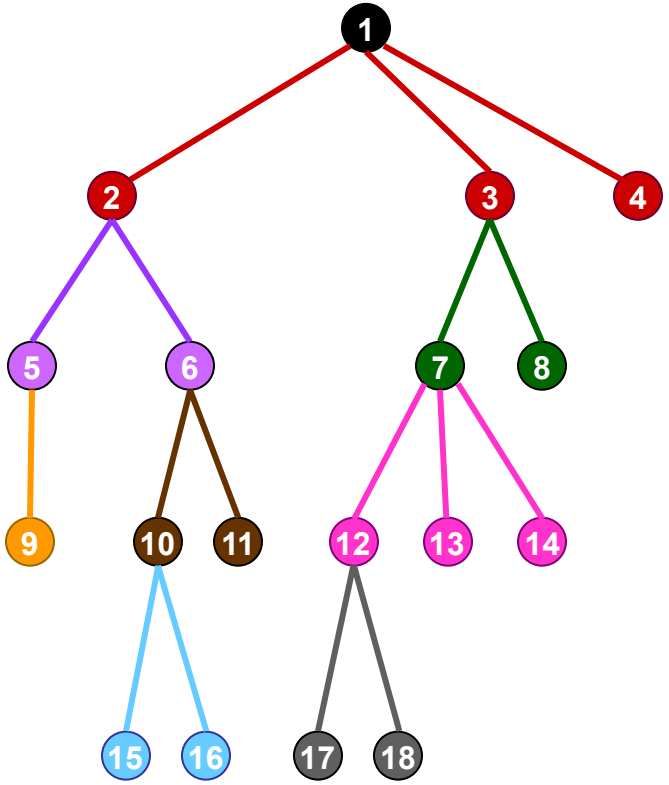



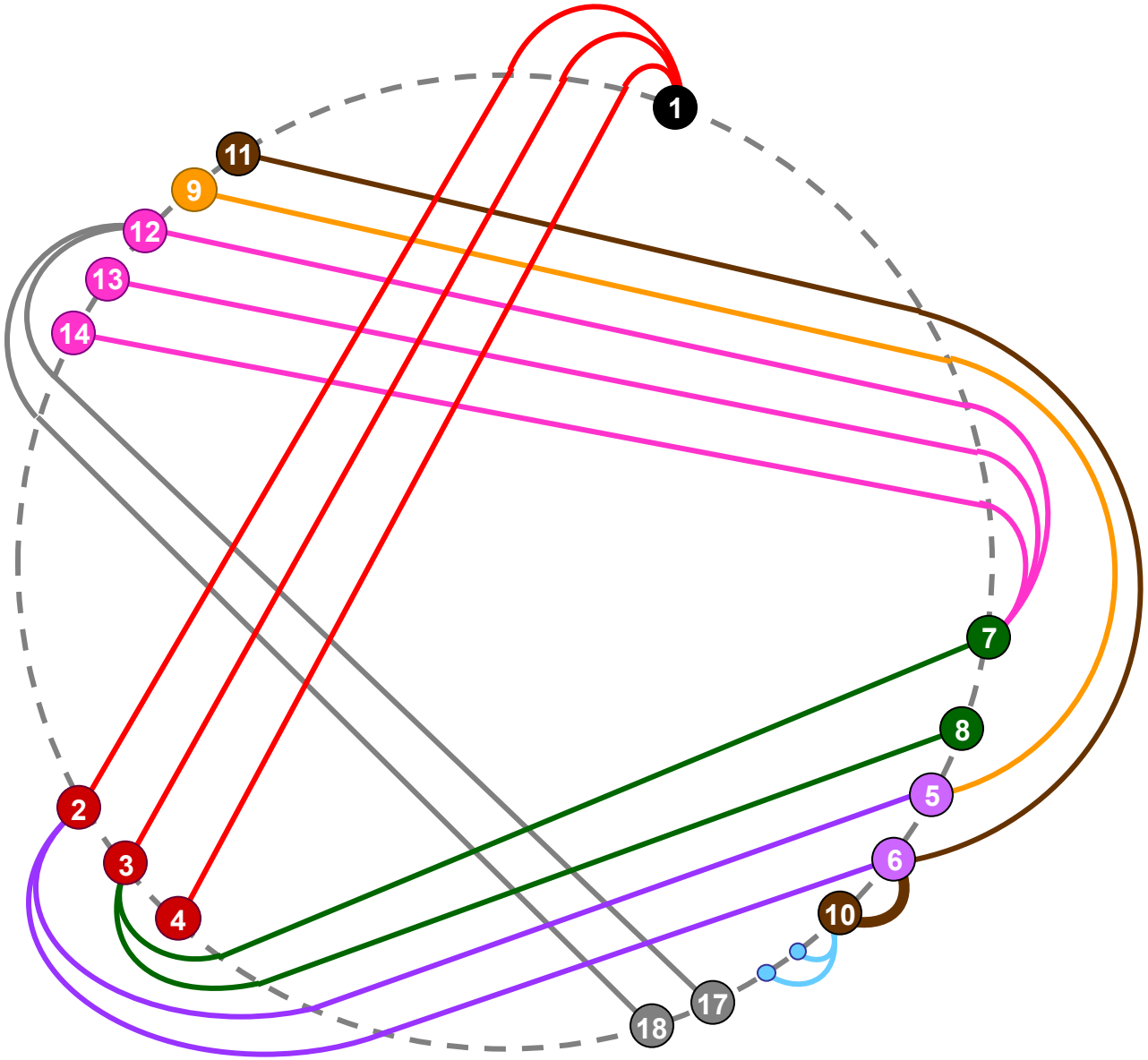
# .....Once every level is a full rainbow....



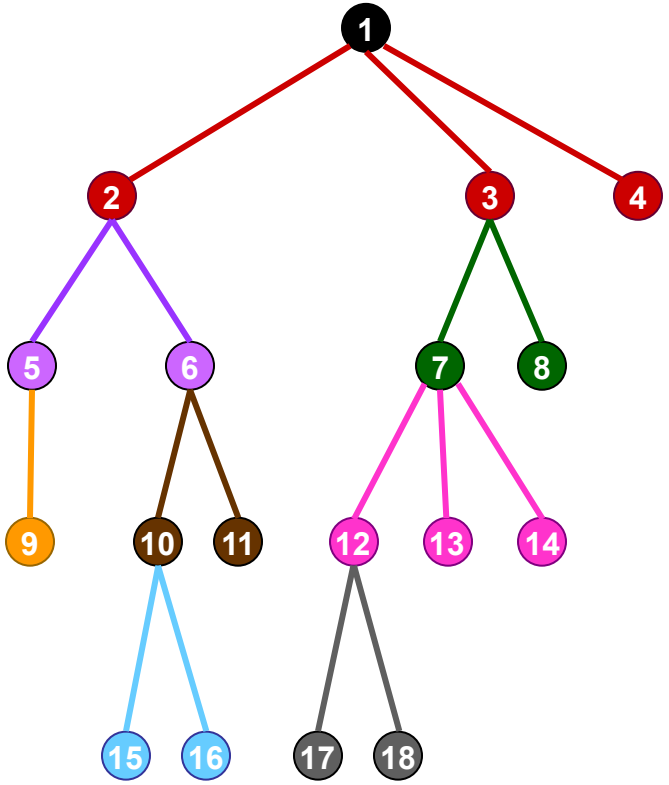



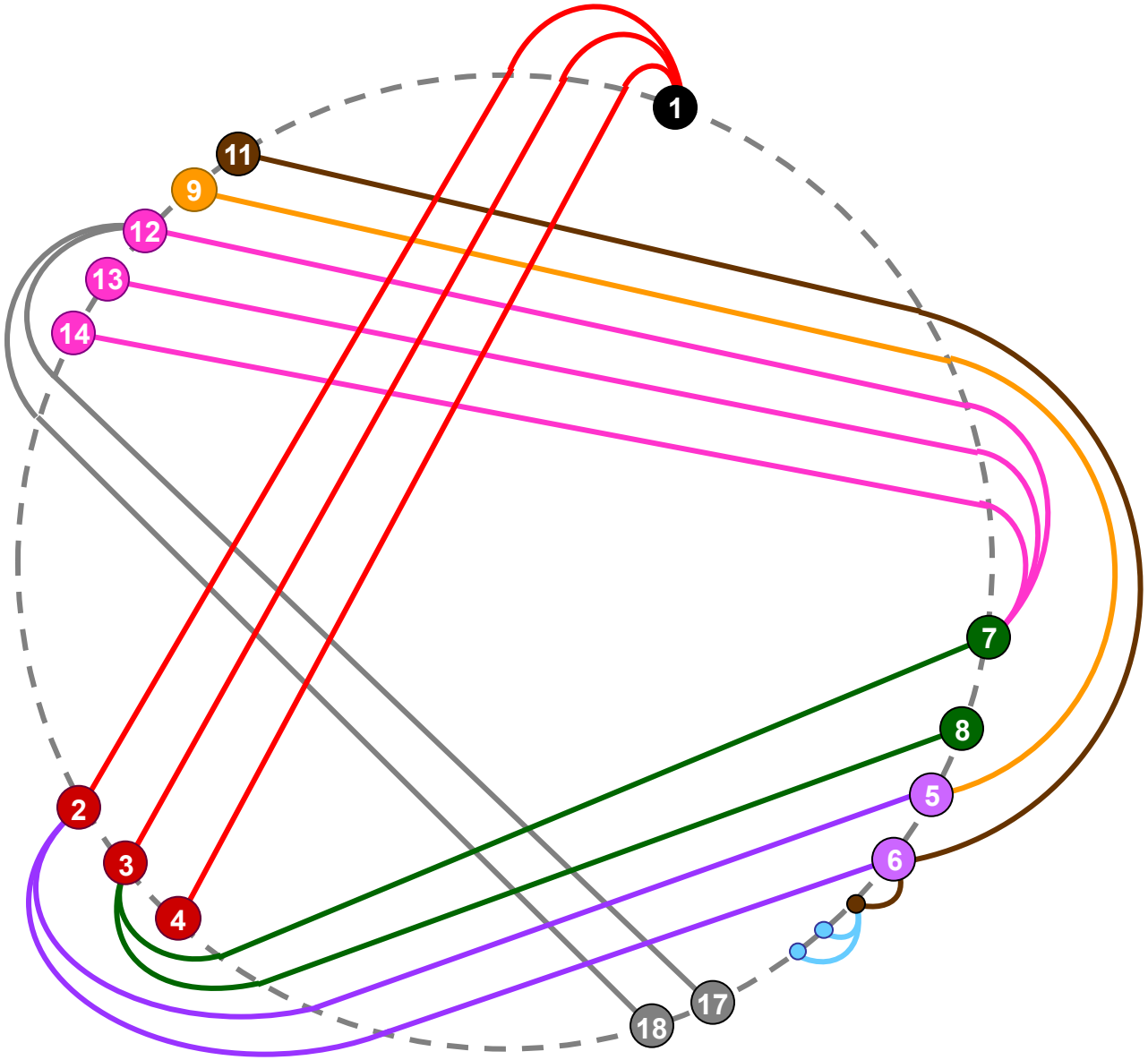
# .....Once every level is a full rainbow....



# .....Once every level is a full rainbow....



# The tangling-untangling theorem

Every T admits a topological circular layout with $\chi$ edge crossings, $\chi \in [0..\vartheta(T)]$, where every edge traverses the spine at most twice. The circular layout can be computed in $O(n^3)$ time

# The prune-tangle-untangle algorithm

# From cubic to quadratic

- **Key obs**: every edge *e* incident to a leaf can cross all other edges except those incident to its parent, hence *e* makes at most O(n) crossings

# From cubic to quadratic

- **Key obs**: every edge *e* incident to a leaf can cross all other edges except those incident to its parent, hence *e* makes at most O(n) crossings
- Step1 (**Prune**): remove enough leaves from T until we get a pruned tree T’ whose thrackle number is “far” from the wanted number of crossings by at most n untangling steps

# From cubic to quadratic

- **Key obs**: every edge *e* incident to a leaf can cross all other edges except those incident to its parent, hence *e* makes at most O(n) crossings
- Step1 (**Prune**): remove enough leaves from T until we get a pruned tree T' whose thrackle number is "far" from the wanted number of crossings by at most n untangling steps
- Step 2 (**Draw**): apply the tangle-untangle algorithm to T' so to obtain a circular layout with $\chi$ crossings; extend this circular layout to represent T with no extra crossings and no extra spine traversals

# The prune-tangling-untangling theorem

Every T admits a topological circular layout with $\chi$ edge crossings, $\chi \in [0..\vartheta(T)]$, where every edge traverses the spine at most twice. The circular layout can be computed in $O(n^2)$ time

# Open Problems

- Is there an $o(n^2)$-time algorithm to compute a topological circular layout of a tree with prescribed number of crossings and constant spine traversals?

- Can the curve complexity of RAC point set embeddings of trees with prescribed crossings be smaller than 9?

- Extend the study to graph classes other than trees (e.g. cacti)

Thank you!!!