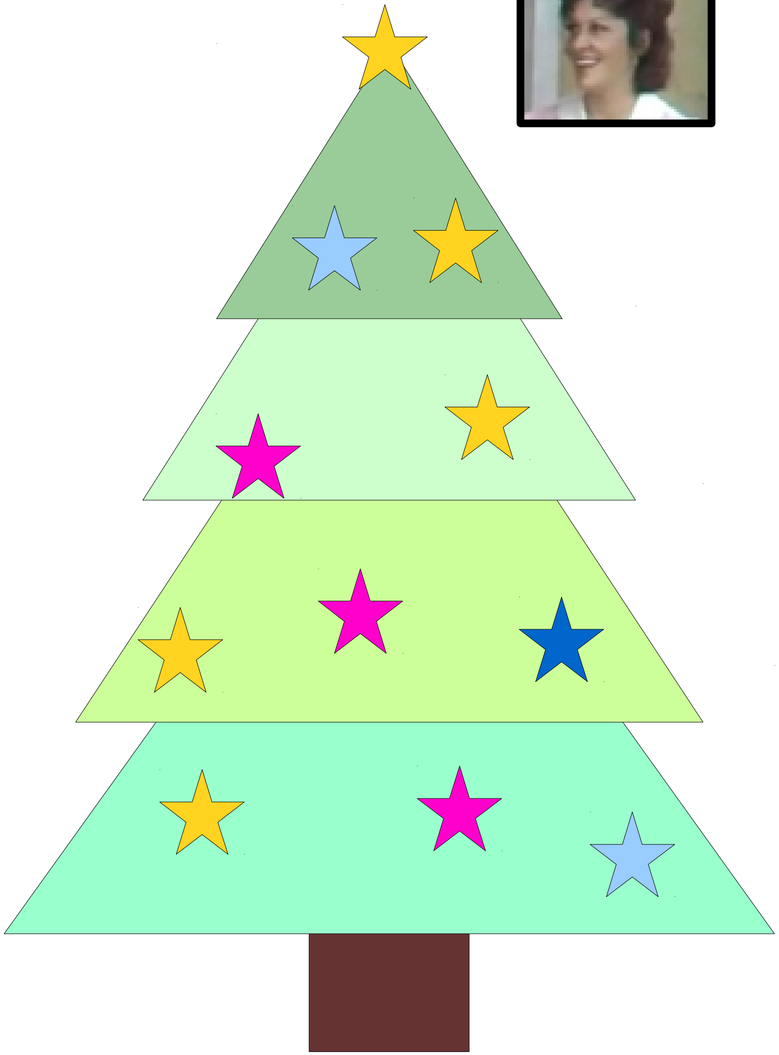
Topcoder SRM 640, D1, 250-Pointer "ChristmasTreeDecoration"

James S. Plank
EECS Department
University of Tennessee

CS494/CS594 Class
August 28, 2018

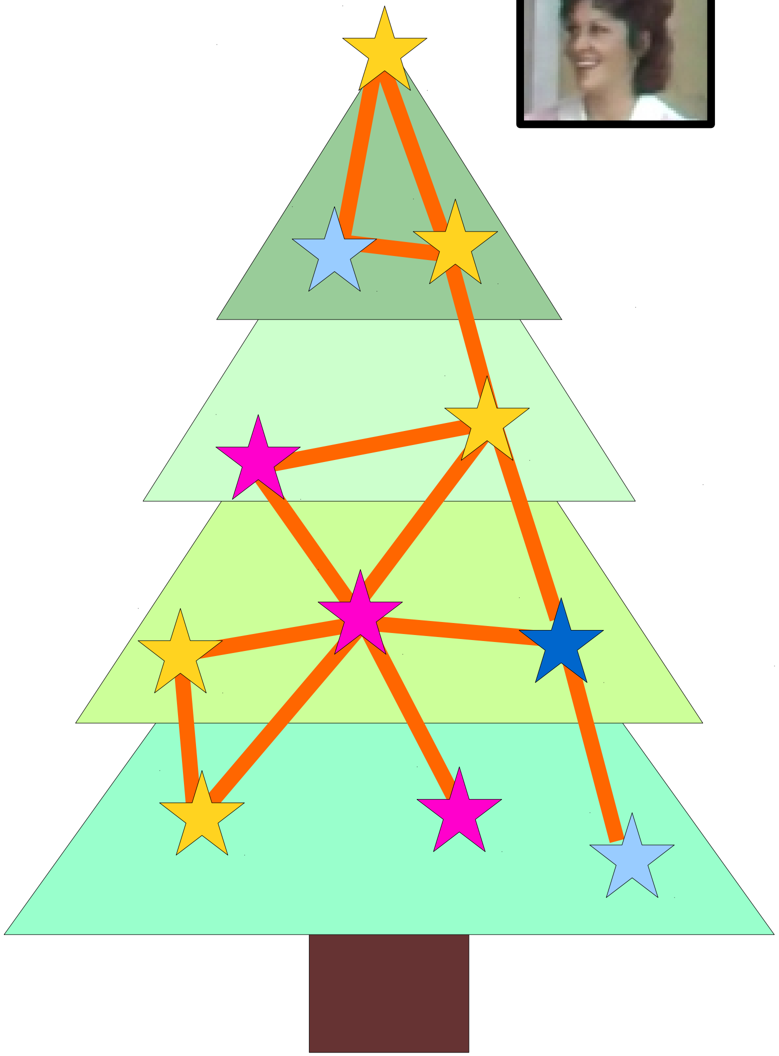
The problem

- Alice has a Christmas tree with *N* colored stars.



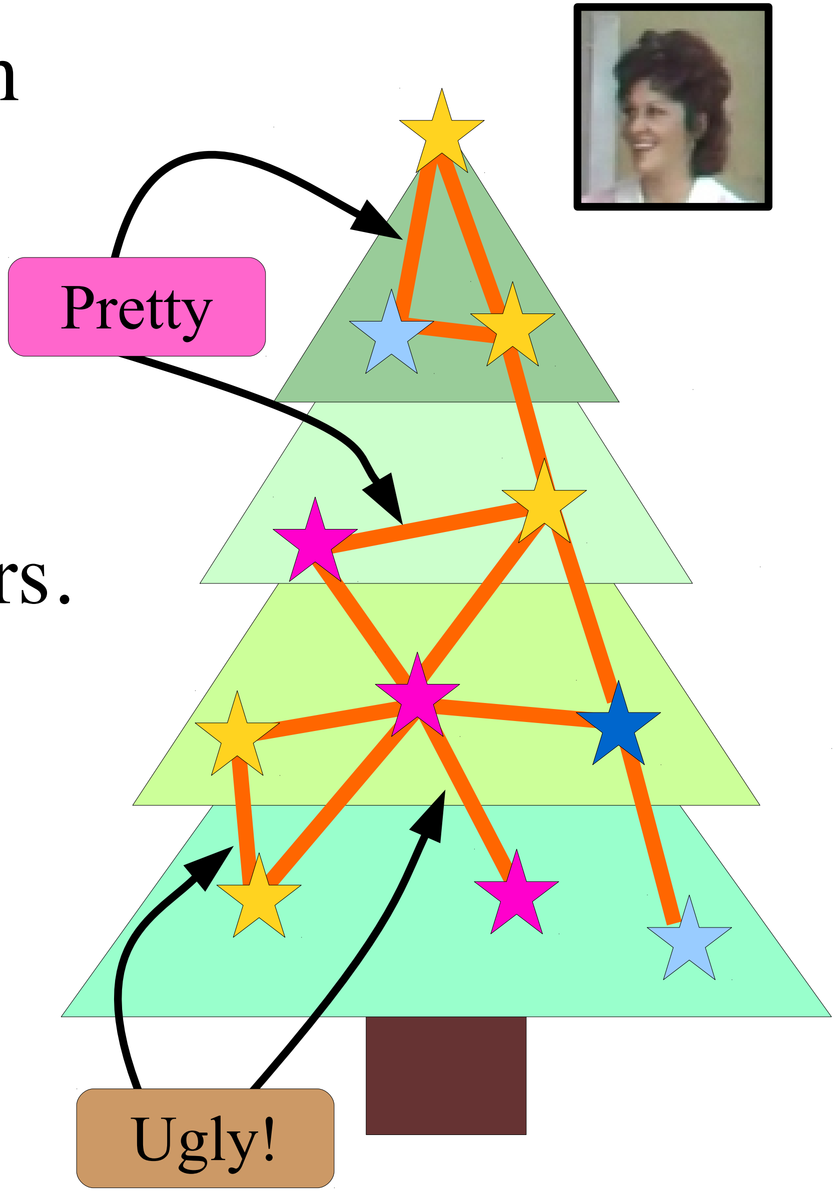
The problem

- Alice has a Christmas tree with *N* colored stars.
- She may tie ribbons between certain stars.



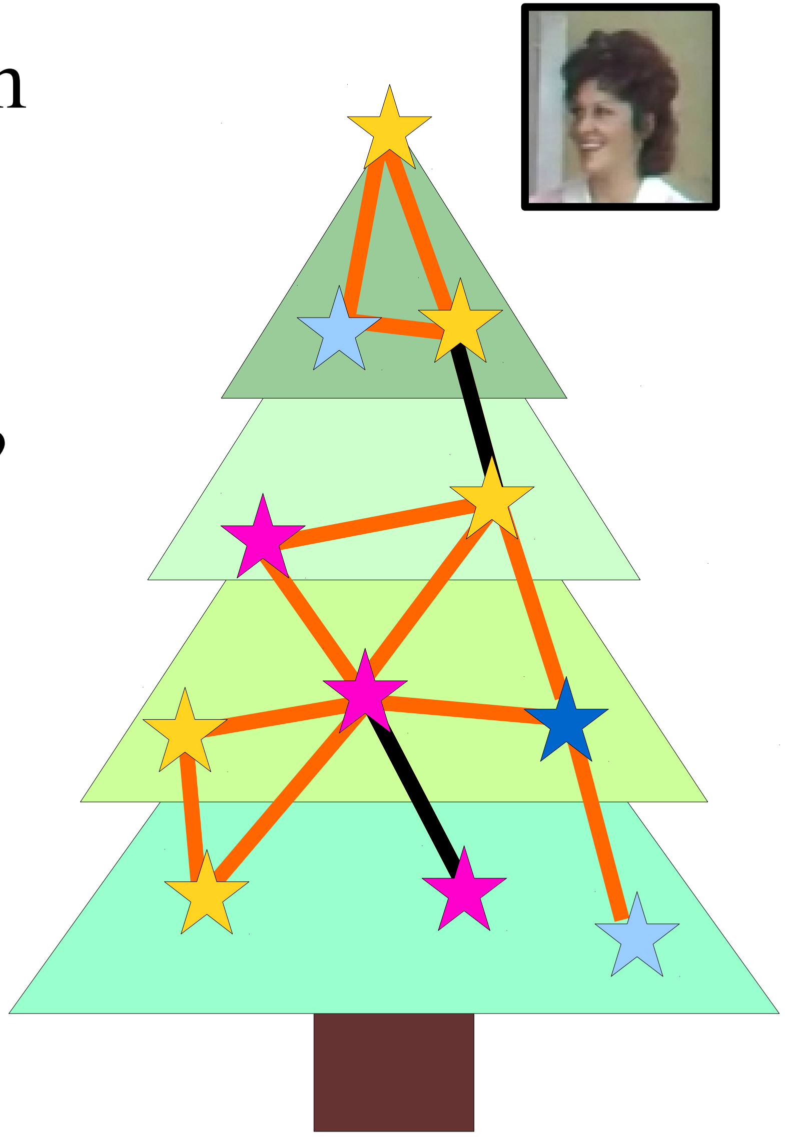
The problem

- Alice has a Christmas tree with *N* colored stars.
- She may tie ribbons between certain stars.
- Ribbons are *pretty* if they connect stars of different colors.



The problem

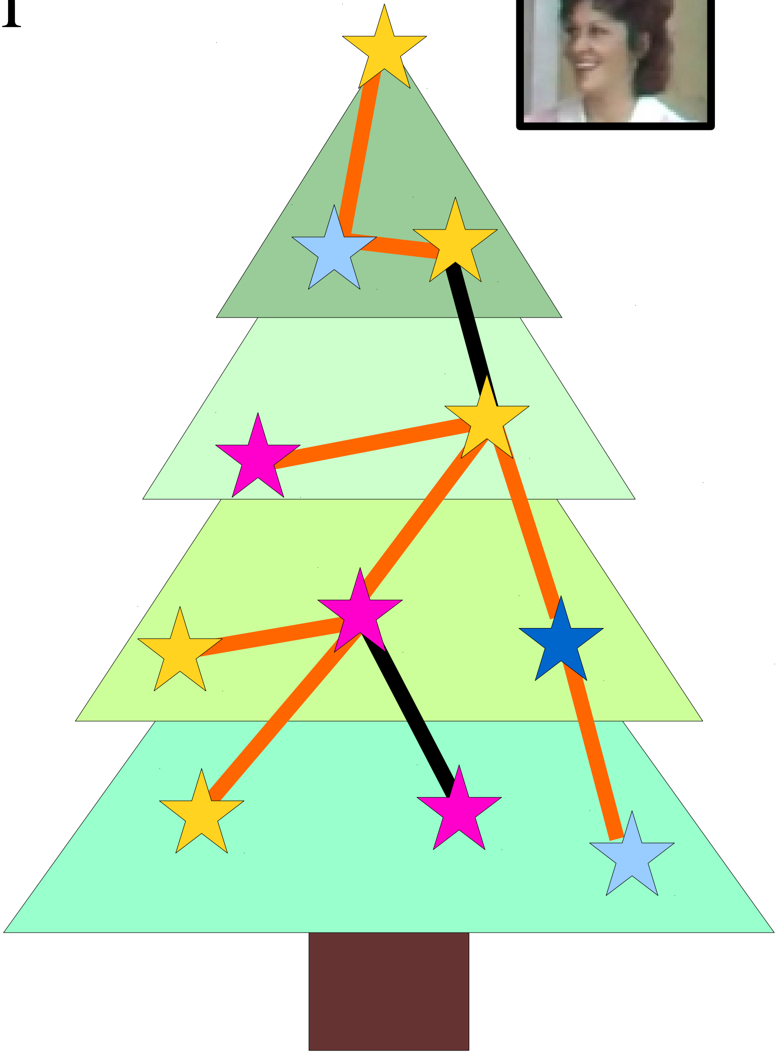
- Alice wants to use the minimum number of ribbons so that all stars are connected.
- What's the minimum number of ugly ribbons that she has to use?



The problem

- Alice wants to use the minimum number of ribbons so that all stars are connected.
- What's the minimum number of ugly ribbons that she has to use?

Two



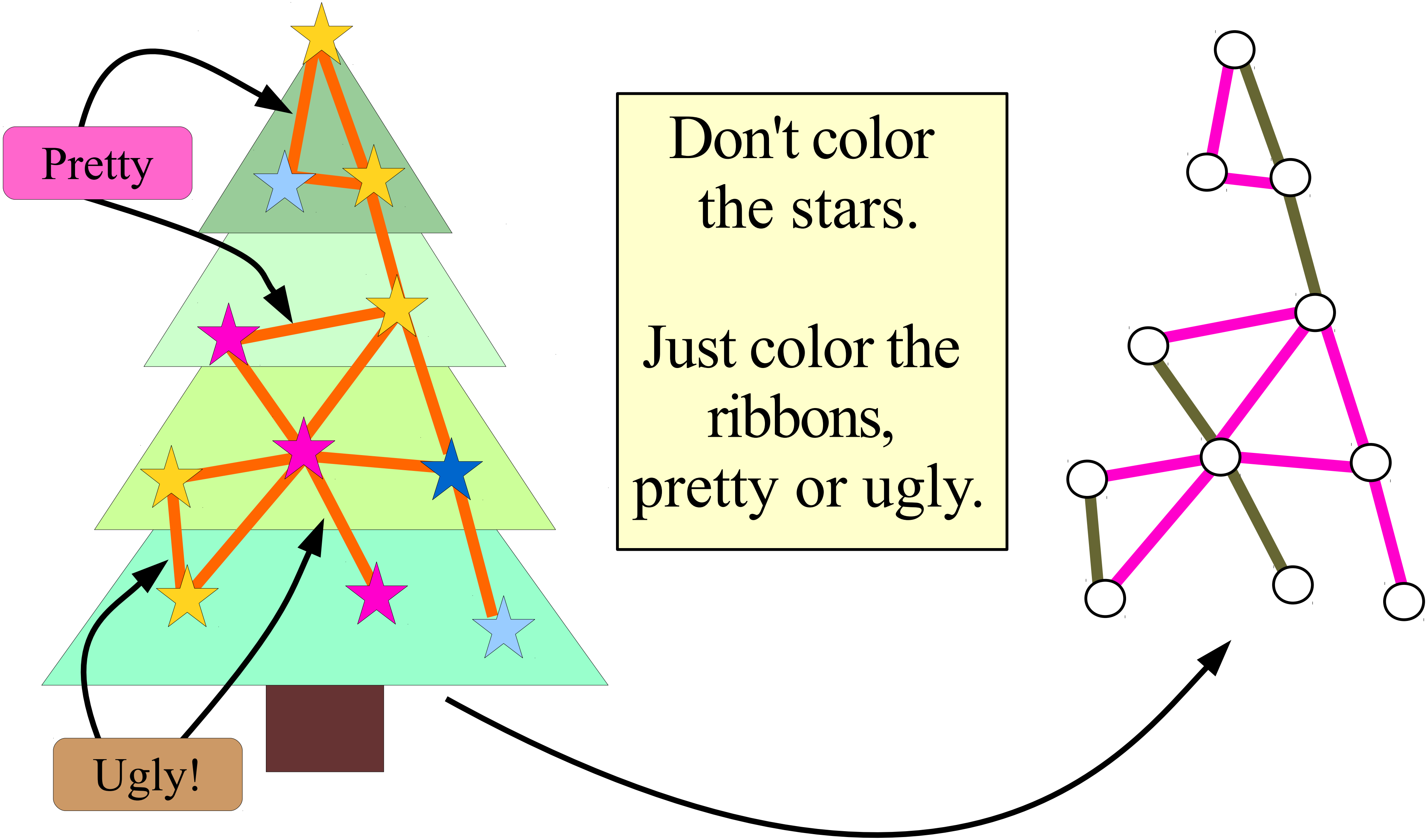
Prototype and Constraints

- **Class name**: `ChristmasTreeDecoration`
- **Method**: `solve()`
- **Parameters**:

col	`vector <int>`	Star colors
x	`vector <int>`	Ribbons – one end
y	`vector <int>`	Ribbons – other end

- **Return Value**: `int`
- **Constraints**:
  - `col.size()`≤ 50.
  - `x.size() == y.size()` ≤ 200.
  - Ribbons guaranteed to connect stars.

Let's View it as a Standard Graph

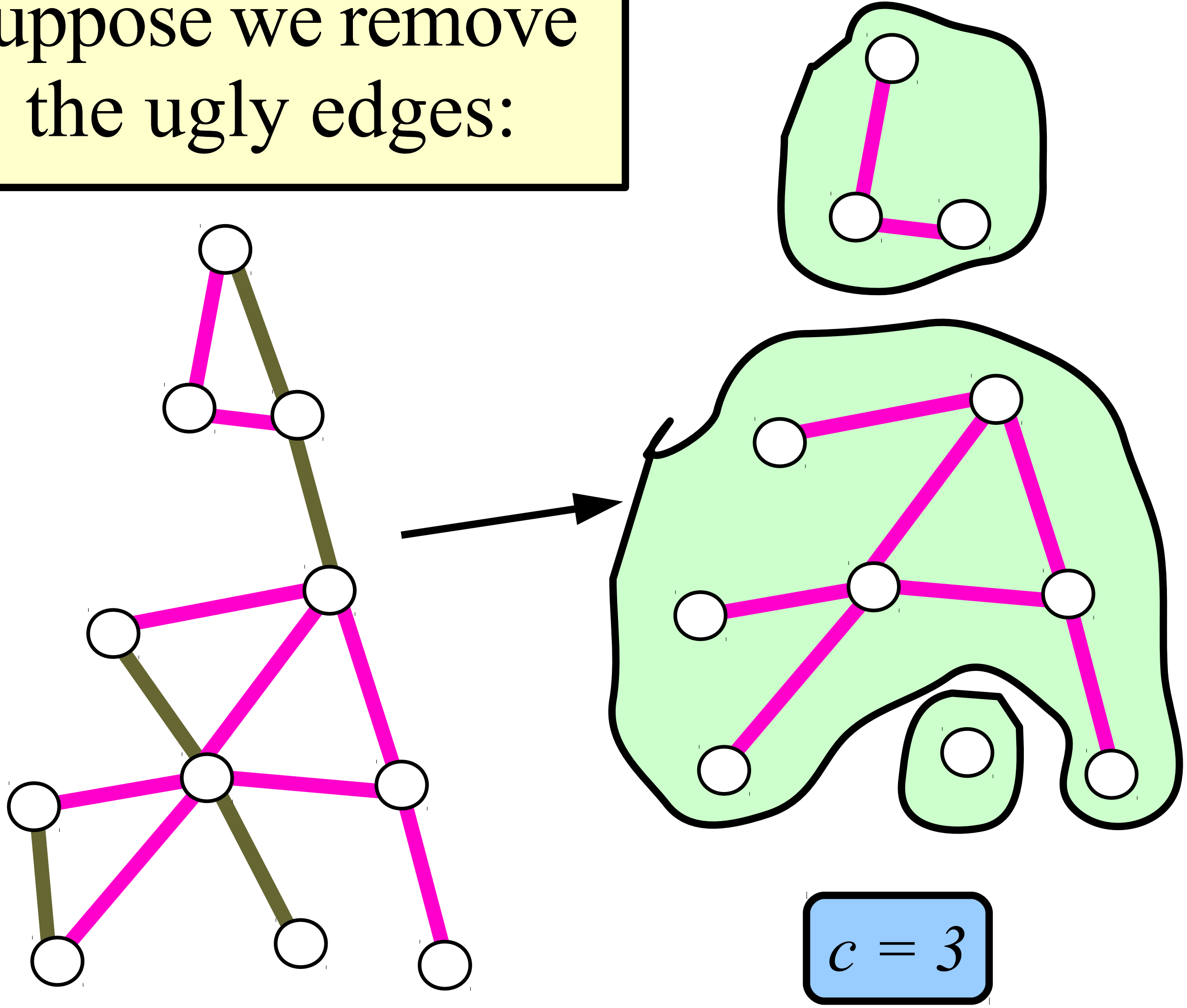


Don't color the stars.

Just color the ribbons, pretty or ugly.

Let's View it as a standard Graph

Suppose we remove the ugly edges:



$c = 3$

We are left with *c* connected components.

We need *(c-1)* ugly edges to connect the components.

The Algorithm:

- Determine the number of connected components, *c*, when only “pretty” ribbons are considered.
- Return *c-1*.

How best to implement it?

- Depth First Search.
- Disjoint Sets.

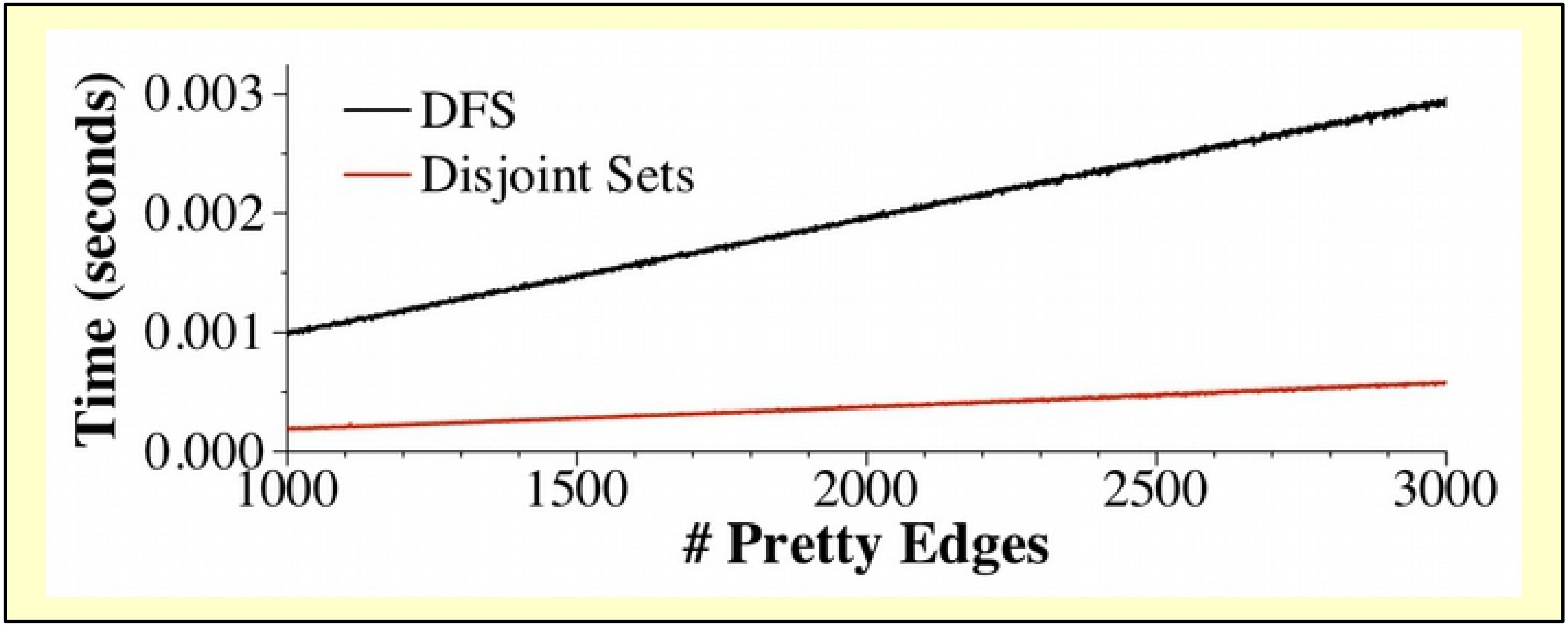
Which is better?

How best to implement it?

- Depth First Search:
  - $O(|V|+|E|)$, which is really $O(|E|)$.
  - Have to create adjacency lists from the list of edges.
  - Then do the recursive DFS.
- Disjoint Sets:
  - $O(|E|\ \alpha(|V|))$, which is really $O(|E|)$.
  - Can work directly on the edges.

Experiment

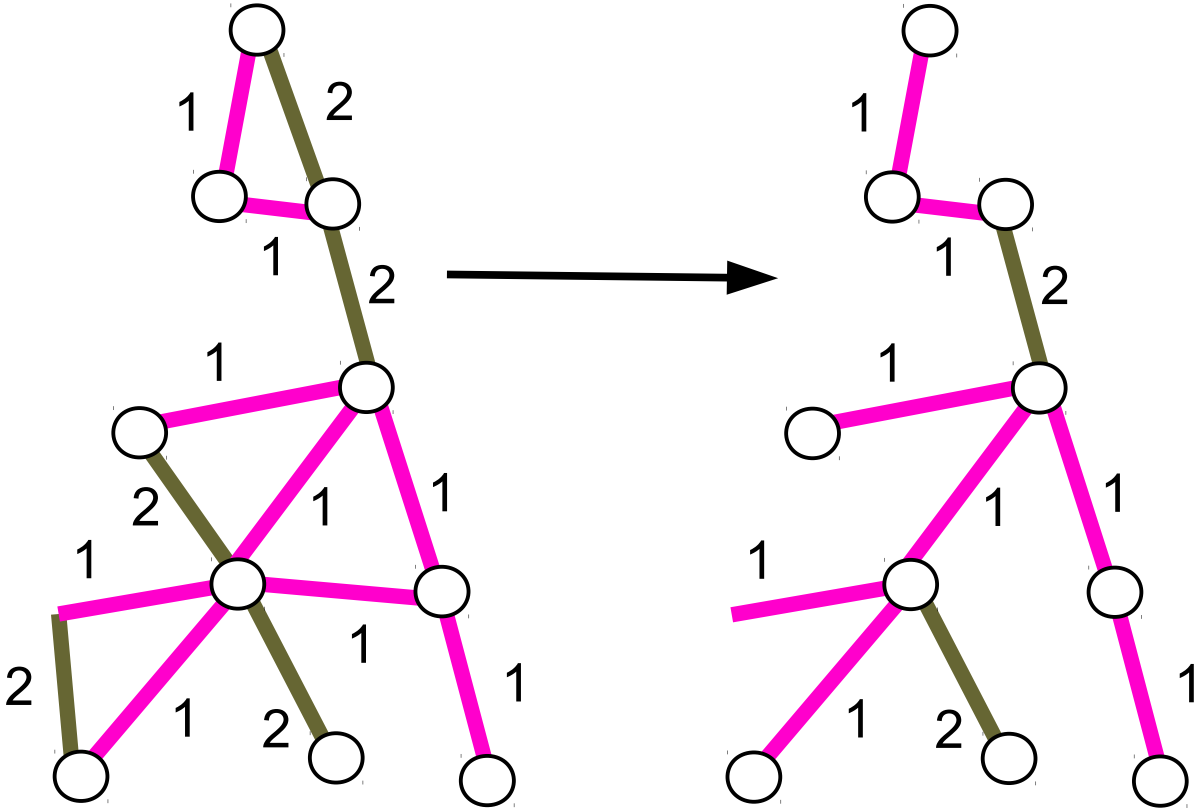
- *N* ranges from 500 to 2000
- 5 colors for the stars – randomly generated
- *2N* random ribbons (connected).
- Mamba (ancient Linux box on my desk)



What about other graph Algorithms?

Pretty edges = 1
Ugly edges = 2

Find the minimum spanning tree & count ugly edges.

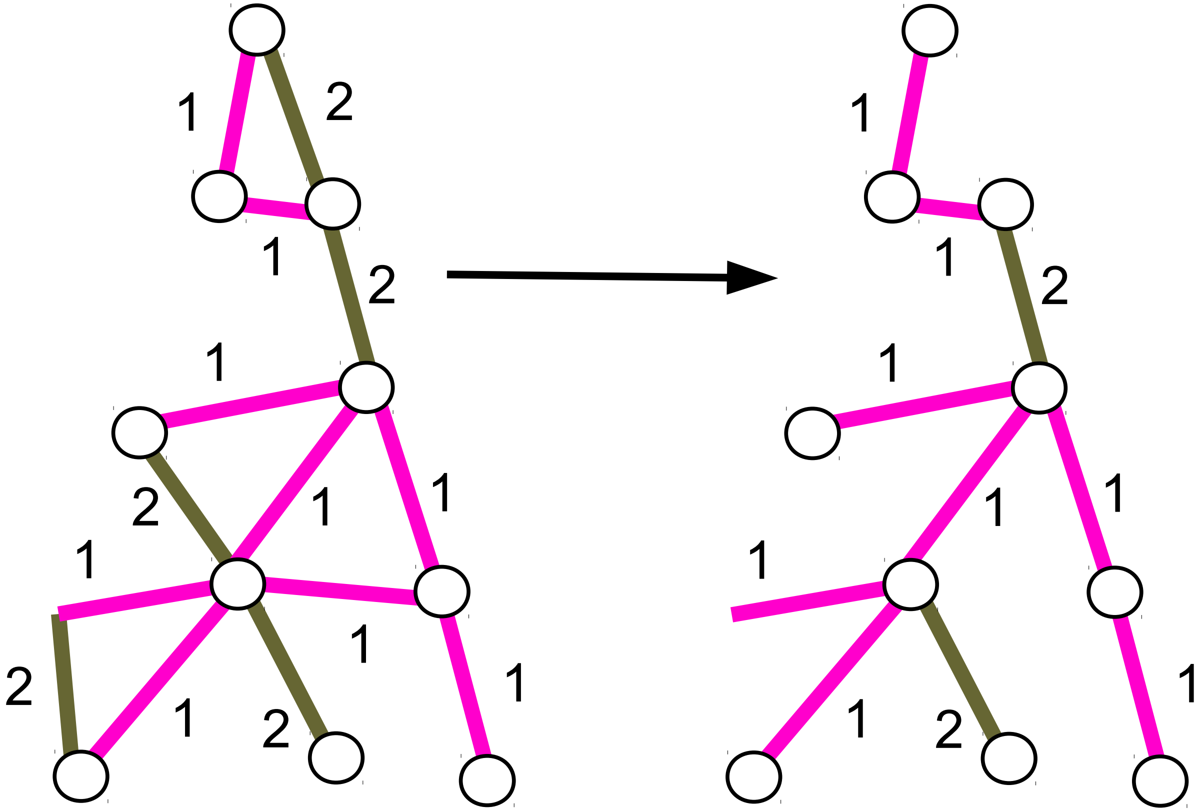


What about other graph Algorithms?

Pretty edges = 1
Ugly edges = 2

Find the minimum spanning tree & count ugly edges.

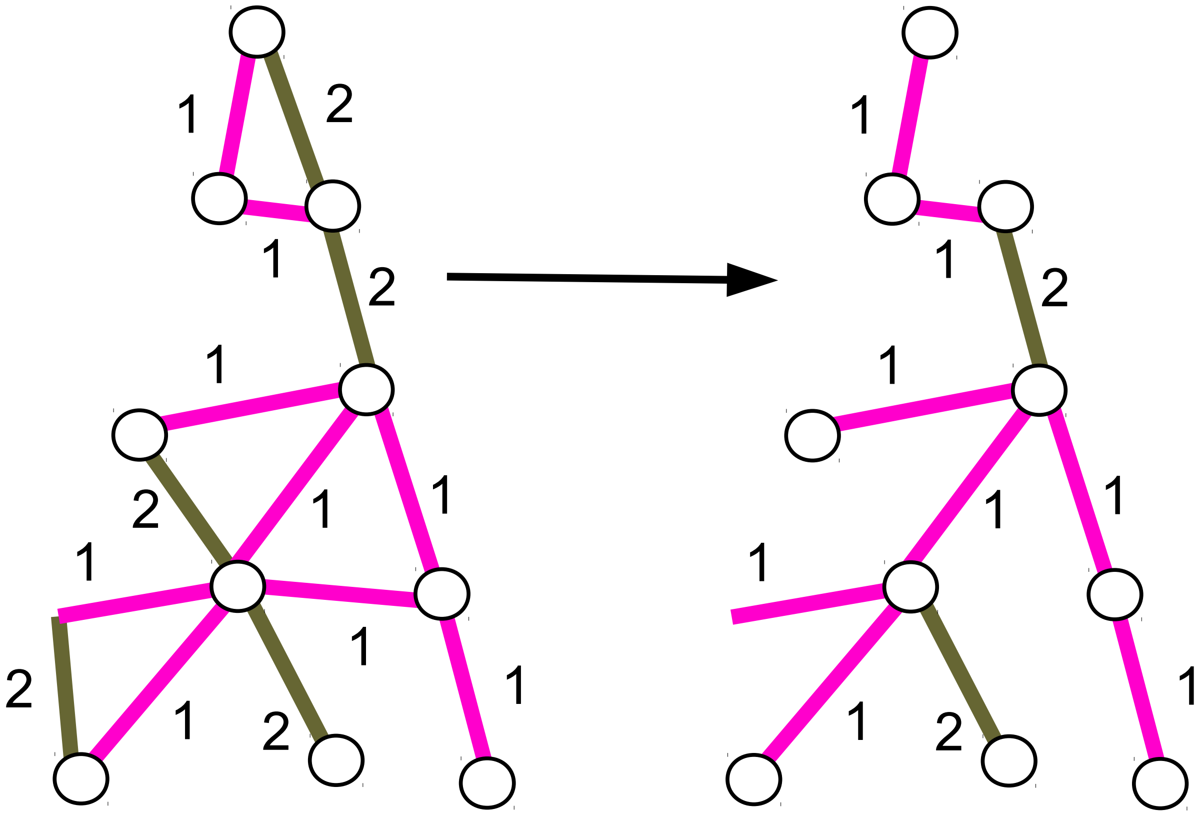
Prim = *O(E)*, because the map can be a linked list (push pretty edges on the front and ugly edges on the back).



What about other graph Algorithms?

Pretty edges = 1
Ugly edges = 2

Find the minimum spanning tree & count ugly edges.



Prim = *O(E)* because the map can be a linked list (push pretty edges on the front and ugly edges on the back).

Kruskal = *O(Eα(V))*

Sorting the edges is *O(E)* because you can use bucket sort.

How did the Topcoders Do?

- This was one of the easier problems:
  - 416 Topcoders opened the problem.
  - 401 (96%) submitted a solution.
  - 365 (91%) of the submissions were correct.
  - Success rate was 87.7%
  - Best time was 2:57
  - Average correct time was 16:04.

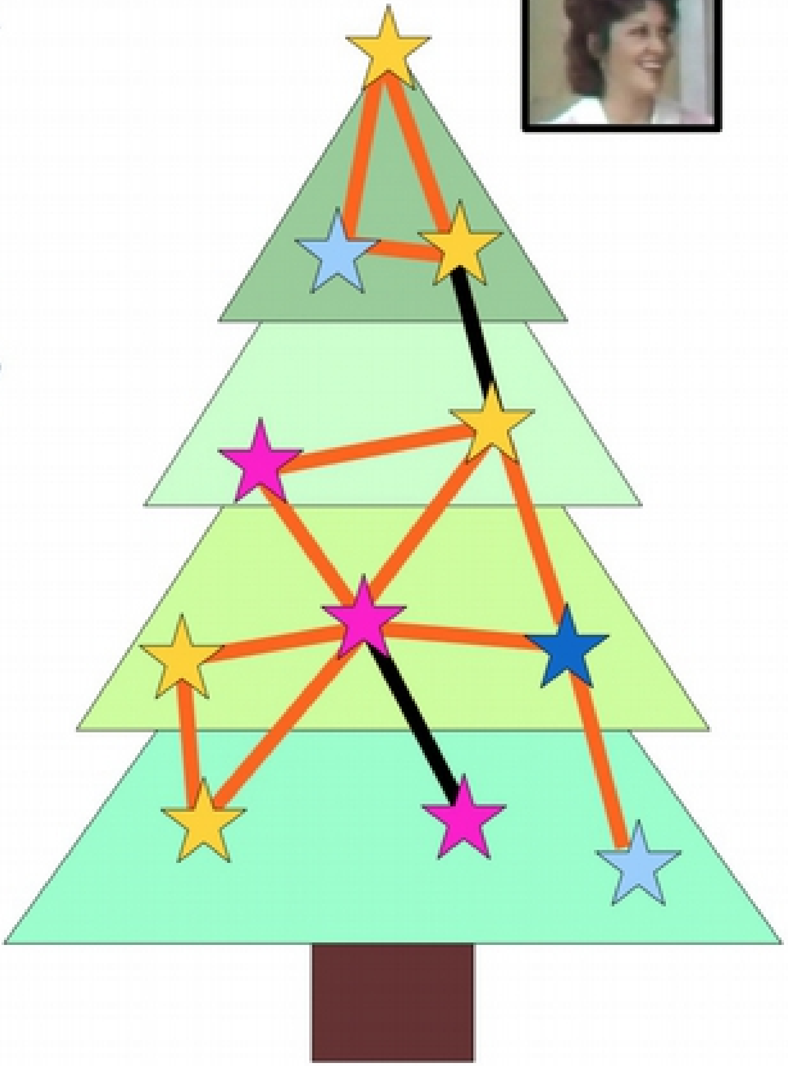
Topcoder SRM 640, D1, 250-Pointer
"ChristmasTreeDecoration"

James S. Plank
EECS Department
University of Tennessee

CS494/CS594 Class
August 28, 2018

The problem

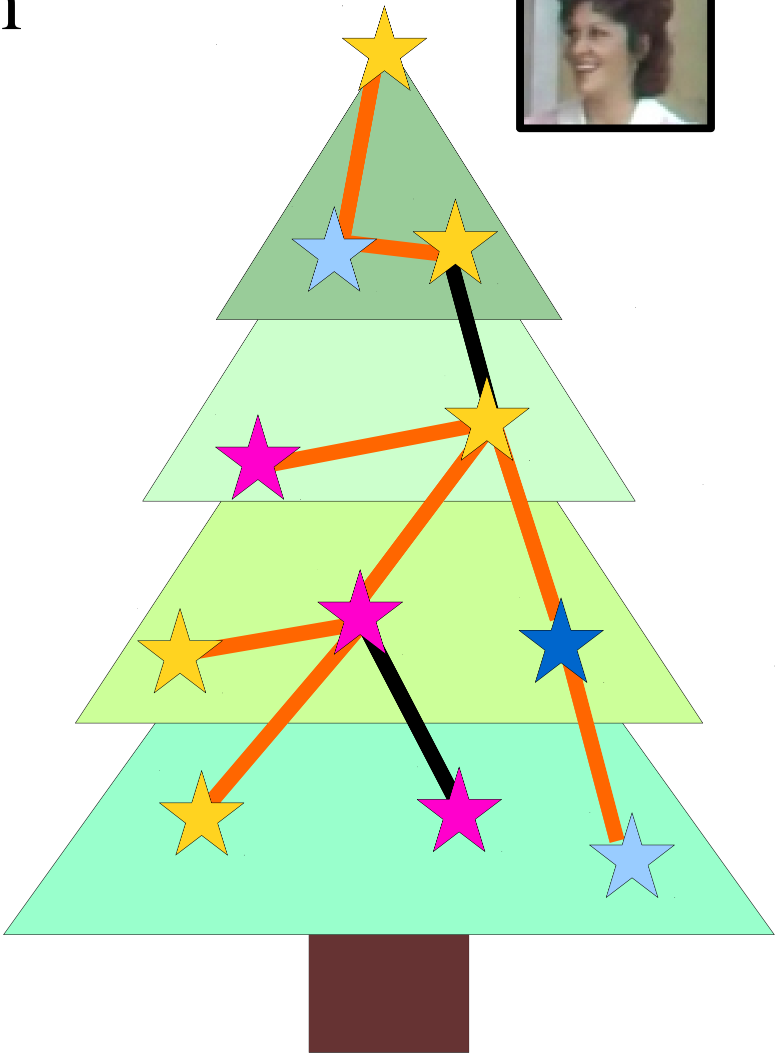
- Alice wants to use the minimum number of ribbons so that all stars are connected.
- What's the minimum number of ugly ribbons that she has to use?



Here's what the animation slide looks like in the PDF

The problem

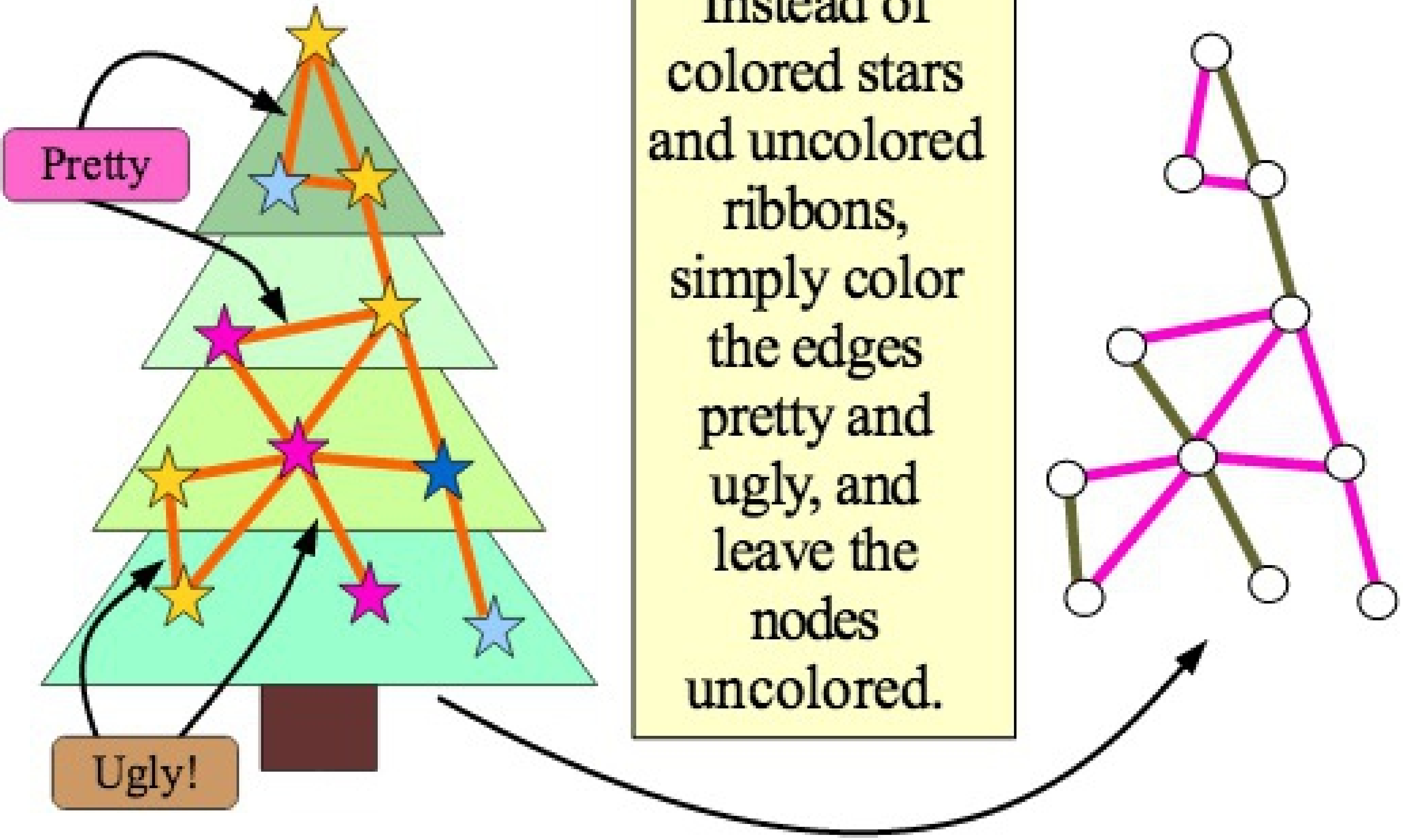
- Alice wants to use the minimum number of ribbons so that all stars are connected.
- What's the minimum number of ugly ribbons that she has to use?



Here's what it should look like.

Before (2015)

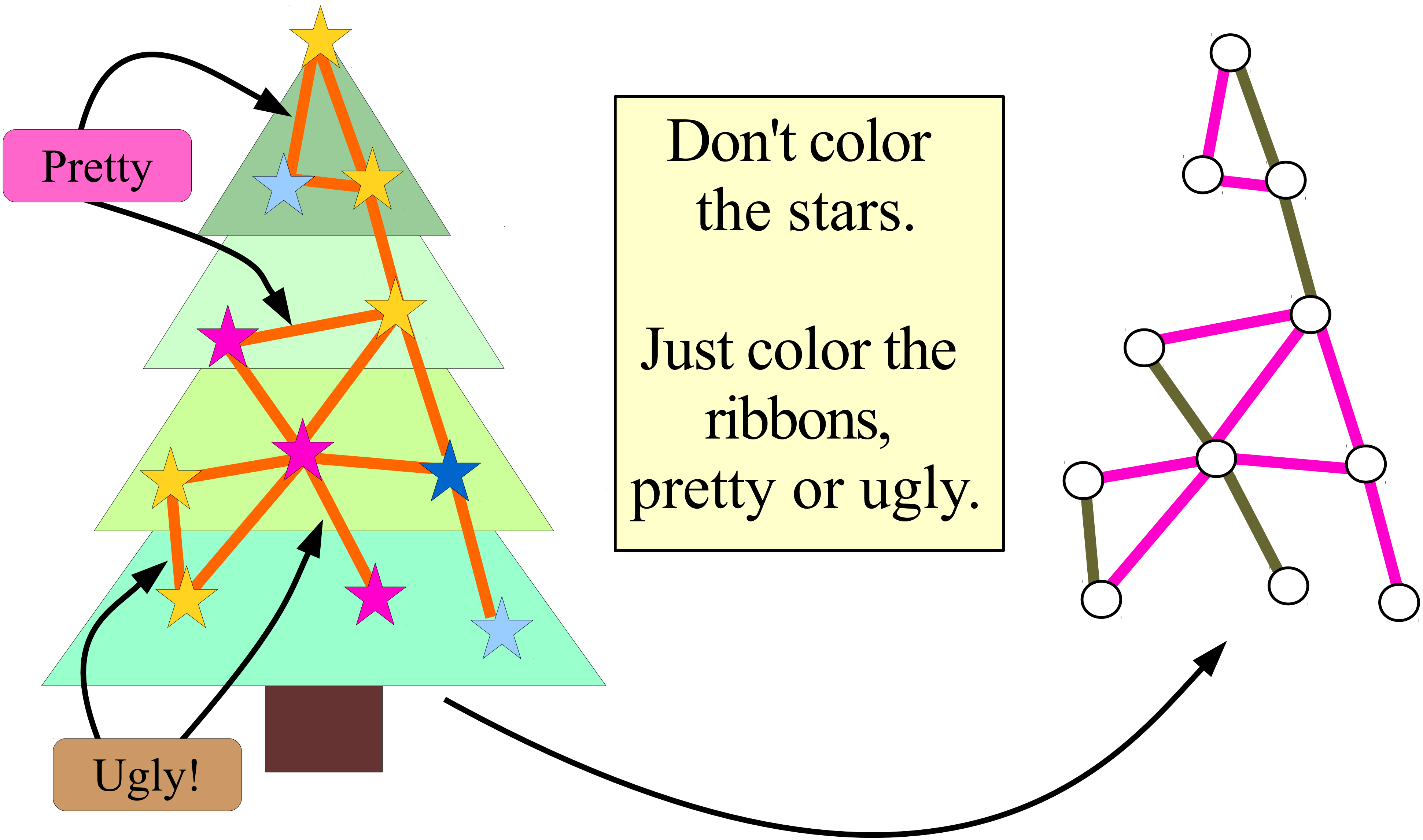
Let's View it as a standard Graph



Instead of colored stars and uncolored ribbons, simply color the edges pretty and ugly, and leave the nodes uncolored.

After

Let's View it as a Standard Graph



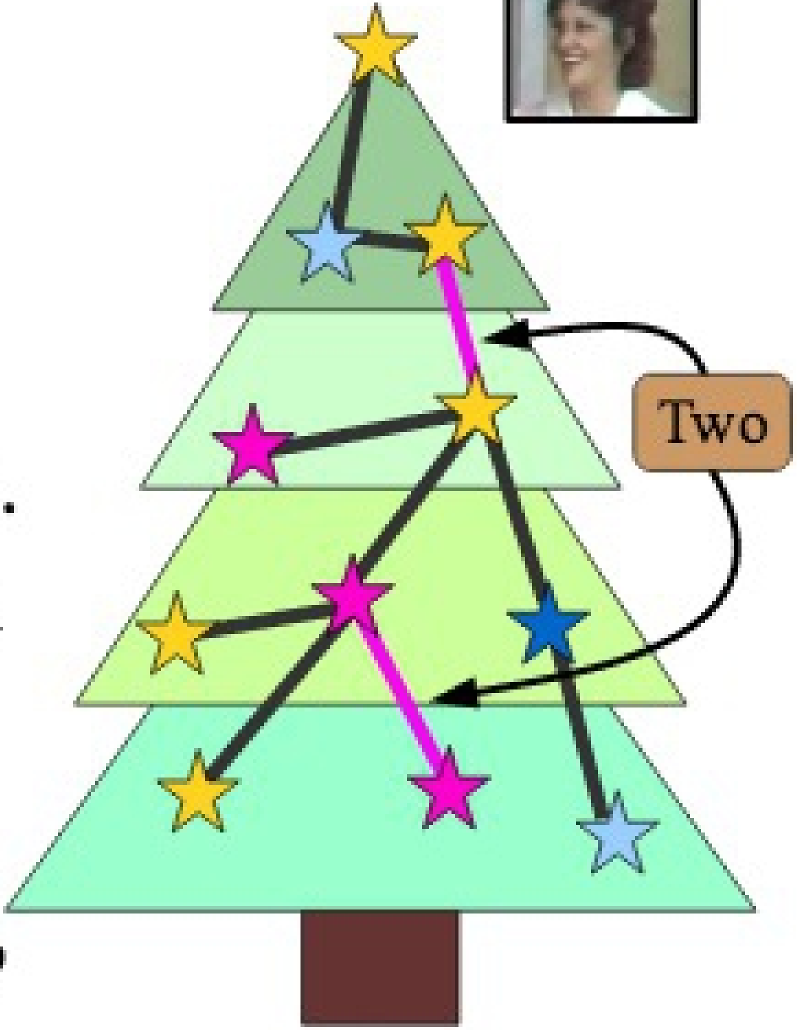
Don't color the stars.

Just color the ribbons, pretty or ugly.

Before (2015)

The problem

- Alice has a Christmas tree with *N* colored stars.
- She may tie ribbons between certain stars.
- Ribbons are *pretty* if they connect stars of different colors.
- Alice wants to use the minimum number of ribbons so that all stars are connected.
- What's the minimum number of ugly ribbons that she has to use?



After

The problem

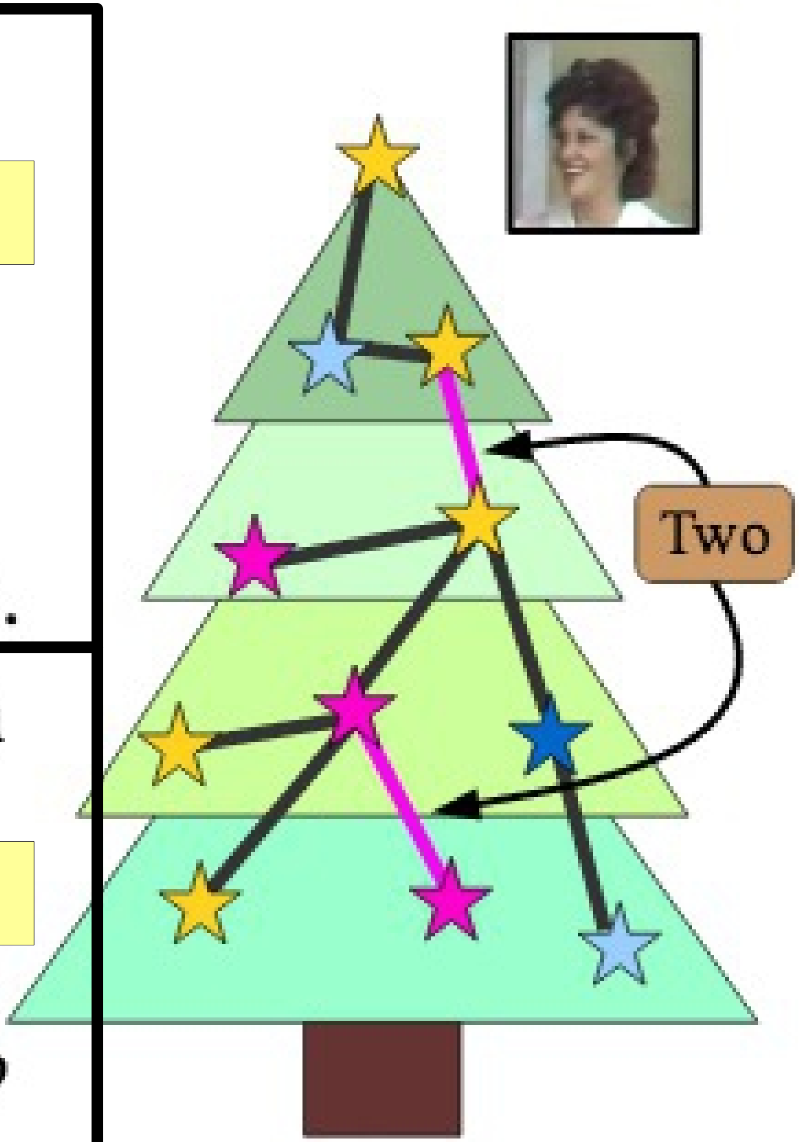
- Alice has a Christmas tree with *N* colored stars.

Slide 1

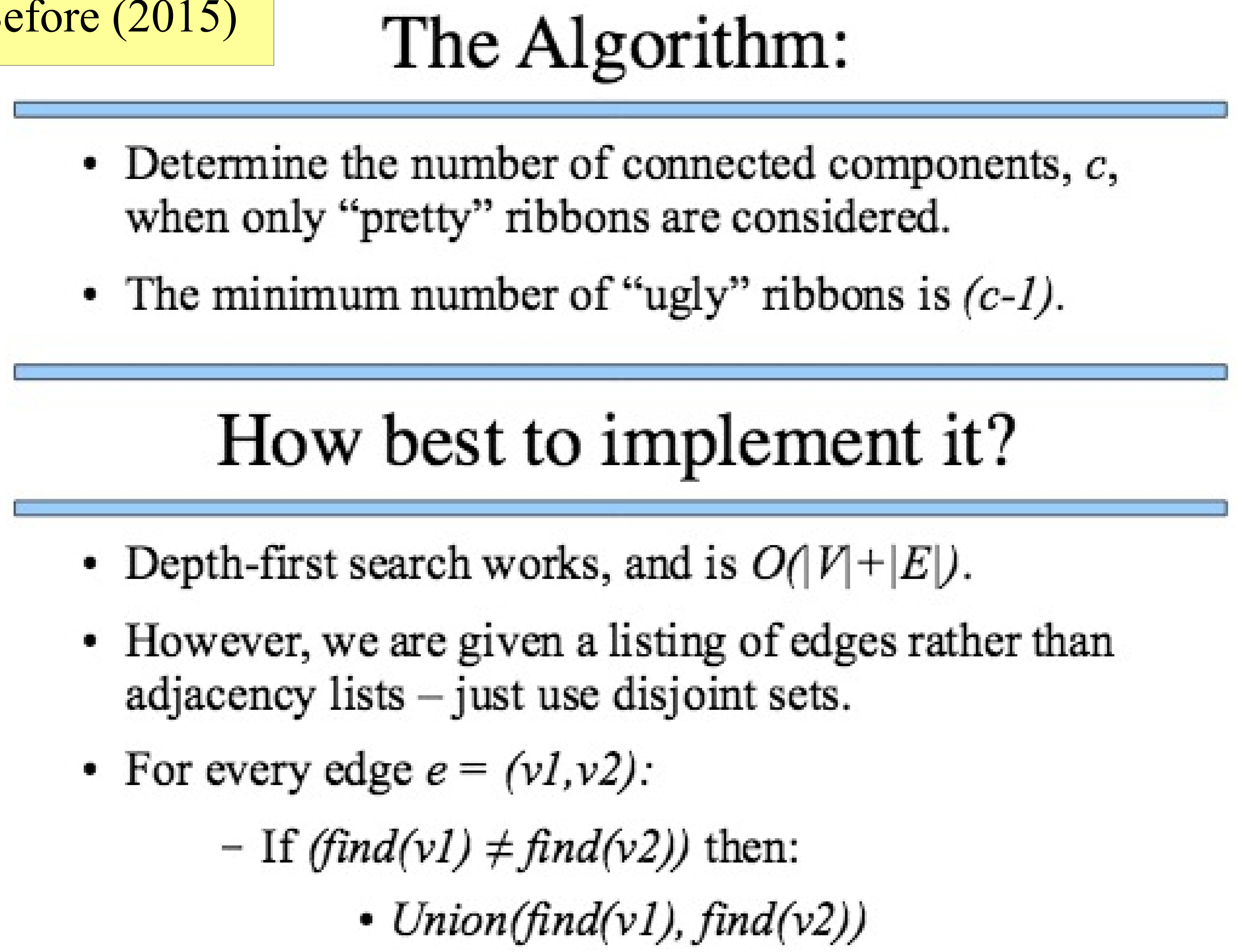
- She may tie ribbons between certain stars.
- Ribbons are *pretty* if they connect stars of different colors.
- Alice wants to use the minimum number of ribbons so that all stars are connected.

Slide 2

- What's the minimum number of ugly ribbons that she has to use?



Before (2015)



The Algorithm:

- Determine the number of connected components, *c*, when only "pretty" ribbons are considered.
- The minimum number of "ugly" ribbons is *(c-1)*.

How best to implement it?

- Depth-first search works, and is $O(|V|+|E|)$.
- However, we are given a listing of edges rather than adjacency lists – just use disjoint sets.
- For every edge $e = (v1,v2)$:
  - If $(find(v1) \neq find(v2))$ then:
    - $Union(find(v1), find(v2))$

After (1)

The Algorithm:

- Determine the number of connected components, *c*, when only “pretty” ribbons are considered.
- Return *c-1*.

How best to implement it?

- Depth First Search.
- Disjoint Sets.

Which is better?

After (2)

ow best to implement it?

- Depth First Search:
  - $O(|V|+|E|)$, which is really $O(|E|)$.
  - Have to create adjacency lists from the list of edges.
  - Then do the recursive DFS.

- Disjoint Sets:
  - $O(|E|\ \alpha(|V|))$, which is really $O(|E|)$.
  - Can work directly on the edges.

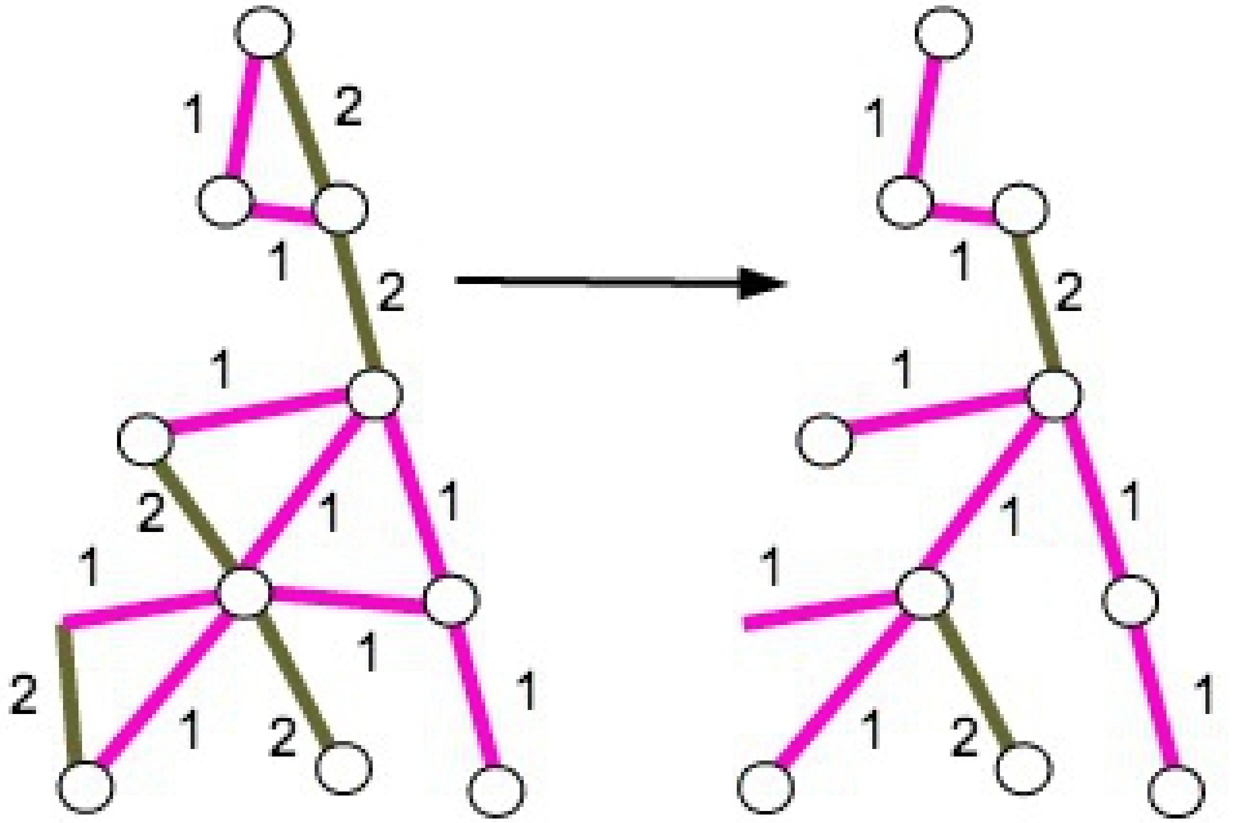
Before (2015)

What about other graph Algorithms?

Pretty edges = 1
Ugly edges = 2

Find the minimum spanning tree & count ugly edges.

Prim = $O(E)$, because the map can be a linked list (push pretty edges on the front and ugly edges on the back).



Kruskal = $O(E\alpha(V))$

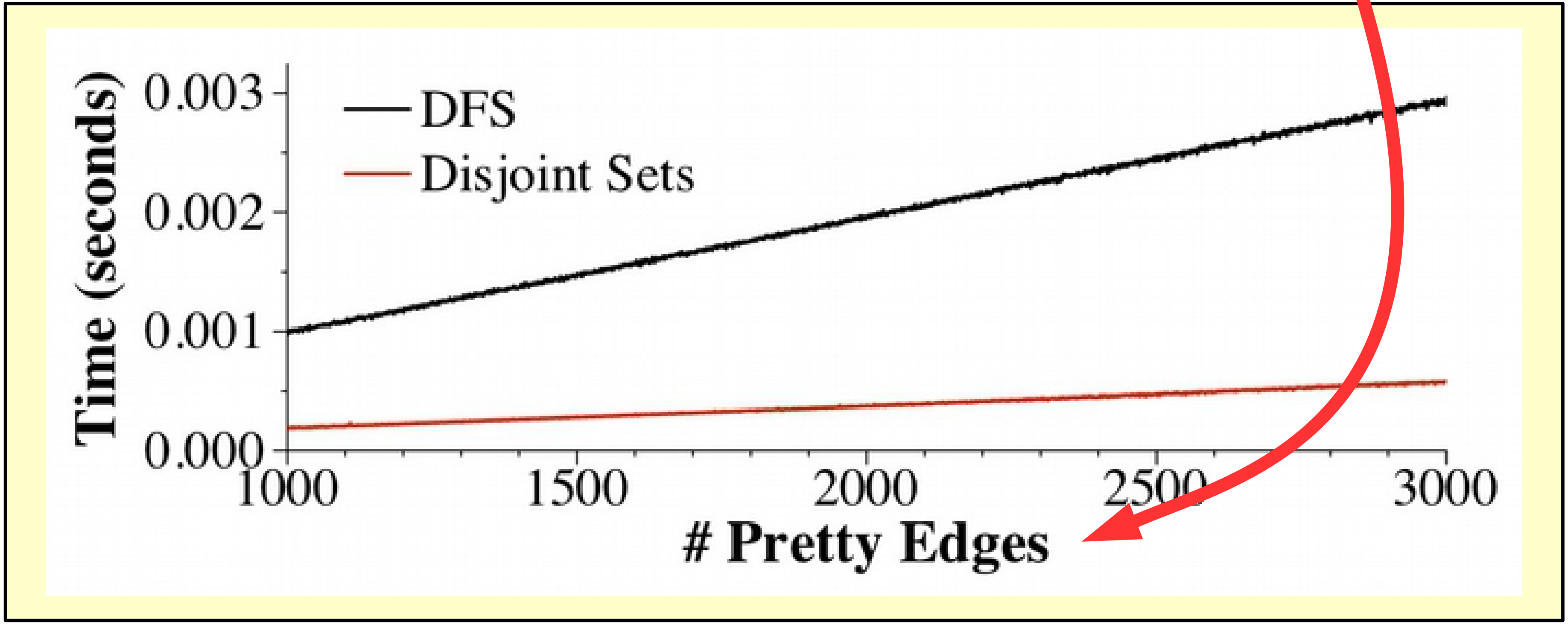
Sorting the edges is $O(E)$ because you can use bucket sort.

What's quirky about this slide?

Experiment

Talk about doing the experiment.

- *N* ranges from 500 to 2000
- 5 colors for the stars – randomly generated
- *2N* random ribbons (connected).
- Mamba (ancient Linux box on my deck)


Time (seconds)
DFS
Disjoint Sets
# Pretty Edges