Dynamic Graphs

Outline for Today

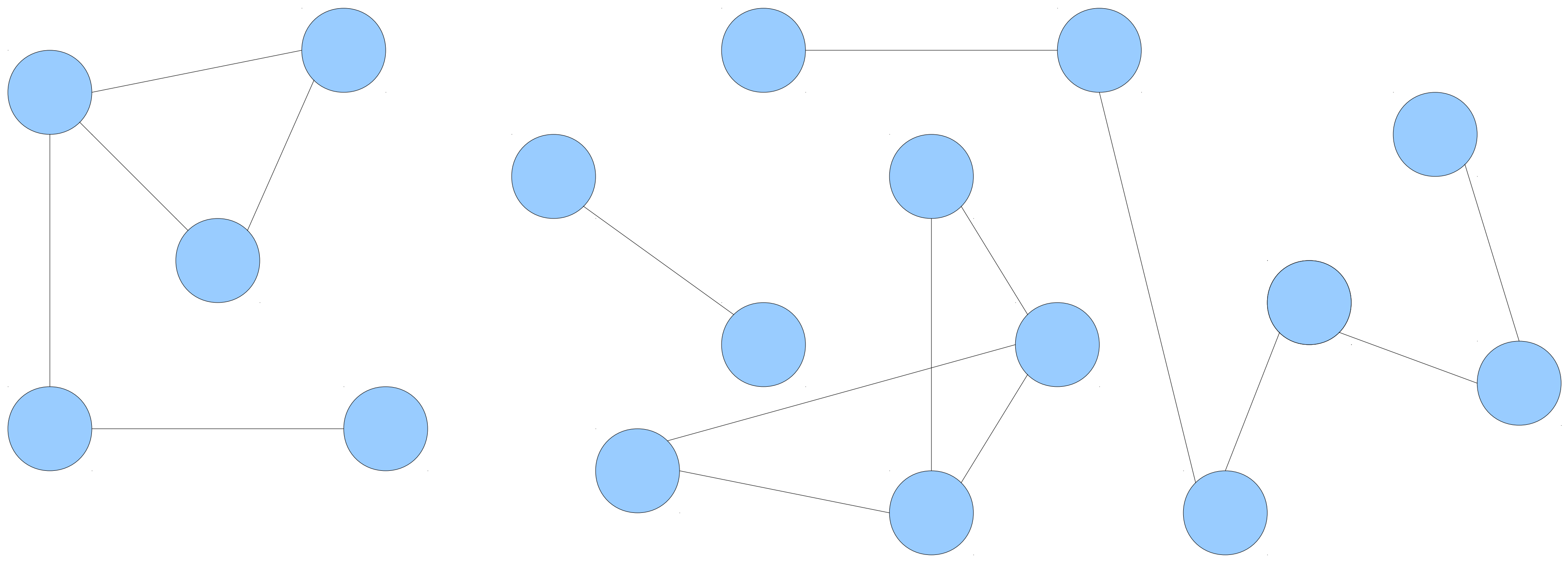
- **Euler Tour Trees Revisited**
  - Dynamic connectivity in forests.
- **The Key Idea**
  - Maintaining dynamic connectivity in general graphs
- **Dynamic Graphs**
  - A data structure for dynamic connectivity.
- **Implementation Details**
  - Speeding up the implementation.

The Dynamic Connectivity Problem

The Connectivity Problem

- The **graph connectivity problem** is the following:

Given an undirected graph *G*, preprocess the graph so that queries of the form “are nodes *u* and *v* connected?”

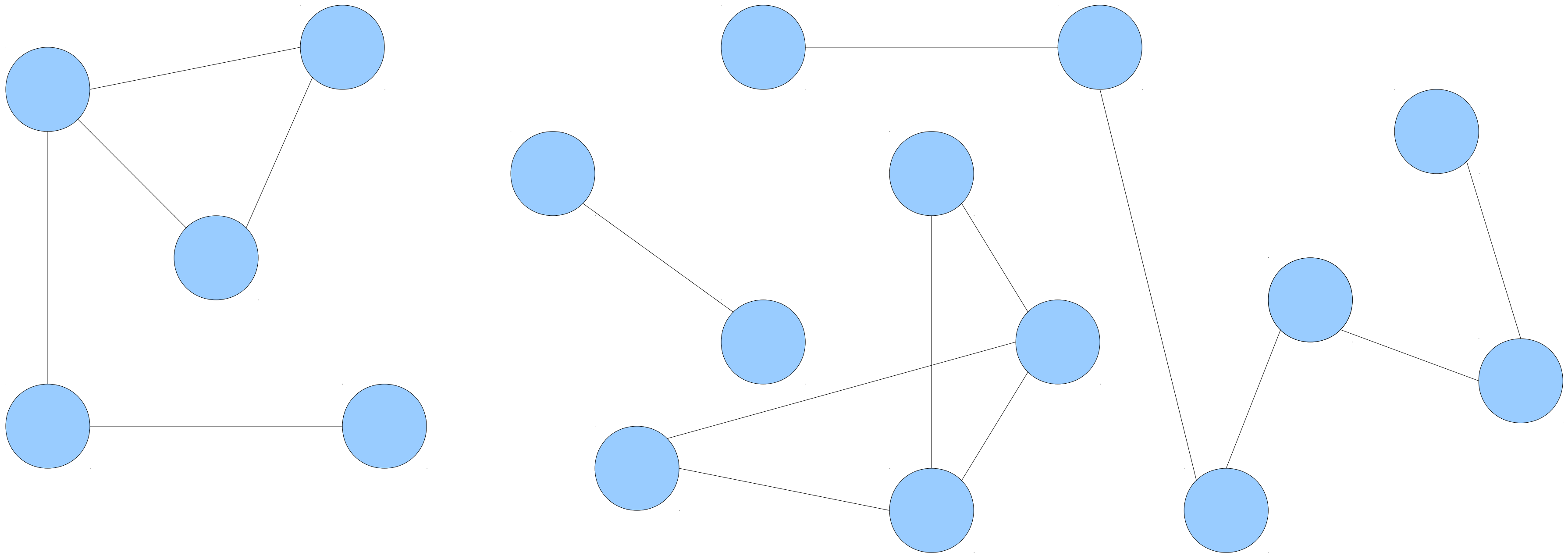


Dynamic Connectivity

- The **dynamic connectivity problem** is the following:

  Maintain an undirected graph *G* so that edges may be inserted an deleted and connectivity queries may be answered efficiently.

- This is a *much* harder problem!



Dynamic Connectivity

- Here's a straightforward solution:
  - Start by doing a DFS to label all nodes with a connected component number.
  - When adding an edge between two nodes in the same CC, do nothing.
  - When adding an edge between two nodes in different CC's, relabel all nodes in the smaller CC.
  - When deleting an edge, rerun DFS on that CC to see if it split.
- Queries run in time O(1); insertions and deletions can take time O(m + n).
- ***Can we do better?***

What We Know

- **Recall:** The **Euler tour tree** data structure solves dynamic connectivity in forests in time O(log n) per query.
- Supports the following operations:
  - ***link***(u, v): Adds edge $\{u, v\}$ to the forest.
  - ***cut***(u, v): Deletes edge $\{u, v\}$ from the forest.
  - ***is-connected***(u, v): Returns whether u and v are connected.
- Implemented on top of balanced BSTs; can augment to compute summary information about each tree.

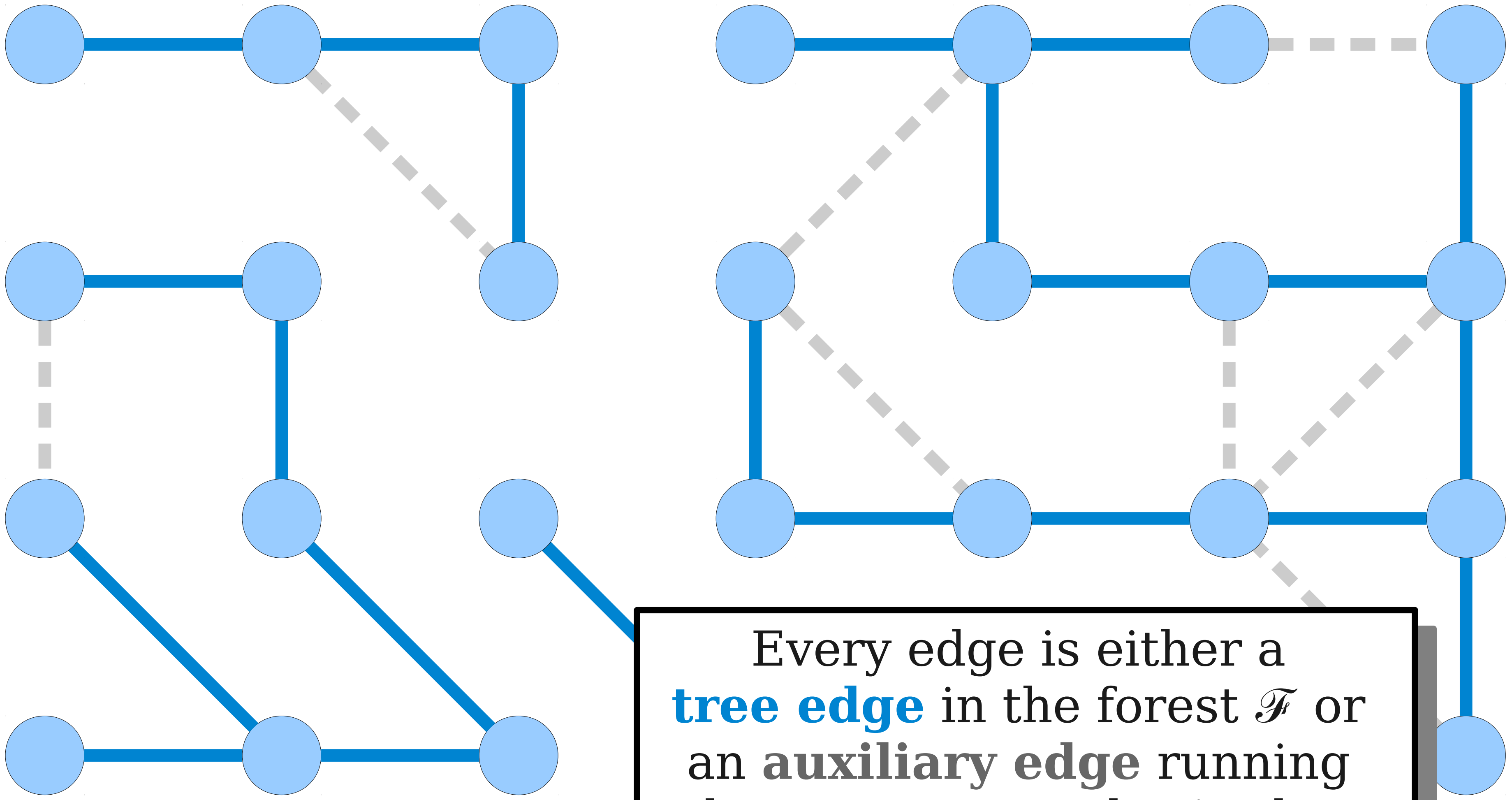
The Challenge

- Numerous issues arise in scaling up from forests to complete graphs.
  - In a forest, a ***link*** connects two distinct trees. In a general graph, the endpoints of a ***link*** might already be connected.
  - In a forest, a ***cut*** splits one tree into two. In a general graph, a ***cut*** might not change connectivity.
  - In a forest, there is a unique path between any two nodes in each tree. In a general graph, there can be many.
- As of 2014, there is no known Euler-tour-like approach for maintaining dynamic connectivity.

The Basic Idea

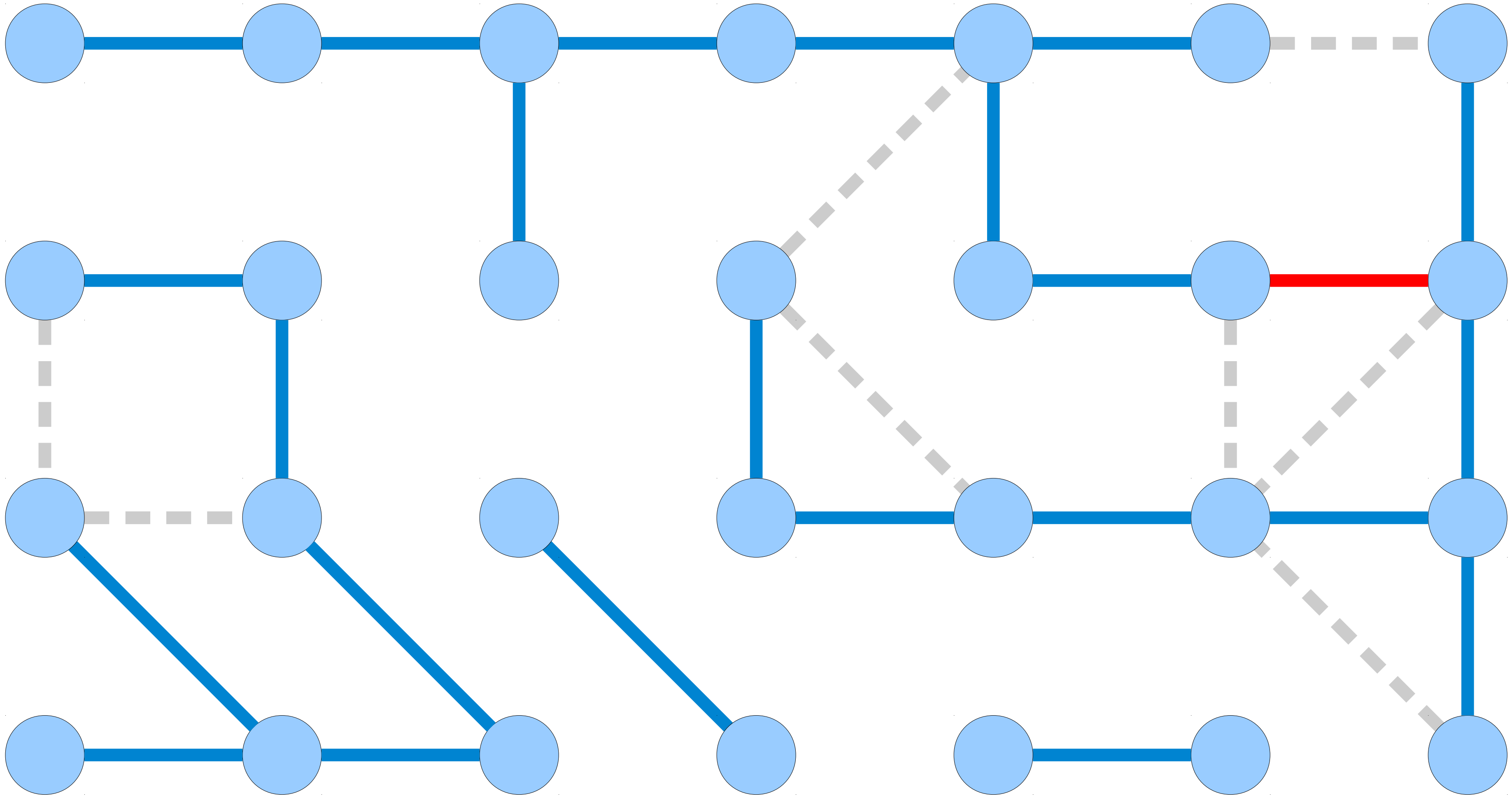
- Let G be an undirected graph and let $\mathscr{F}$ be a spanning forest for G.
- **Observation:** Two nodes u and v are connected in G iff they are connected in $\mathscr{F}$.
- **Idea:** Try to maintain a spanning forest $\mathscr{F}$ for G, represented as Euler tour trees.
- The challenge will be efficiently maintaining $\mathscr{F}$.

Maintaining a Forest

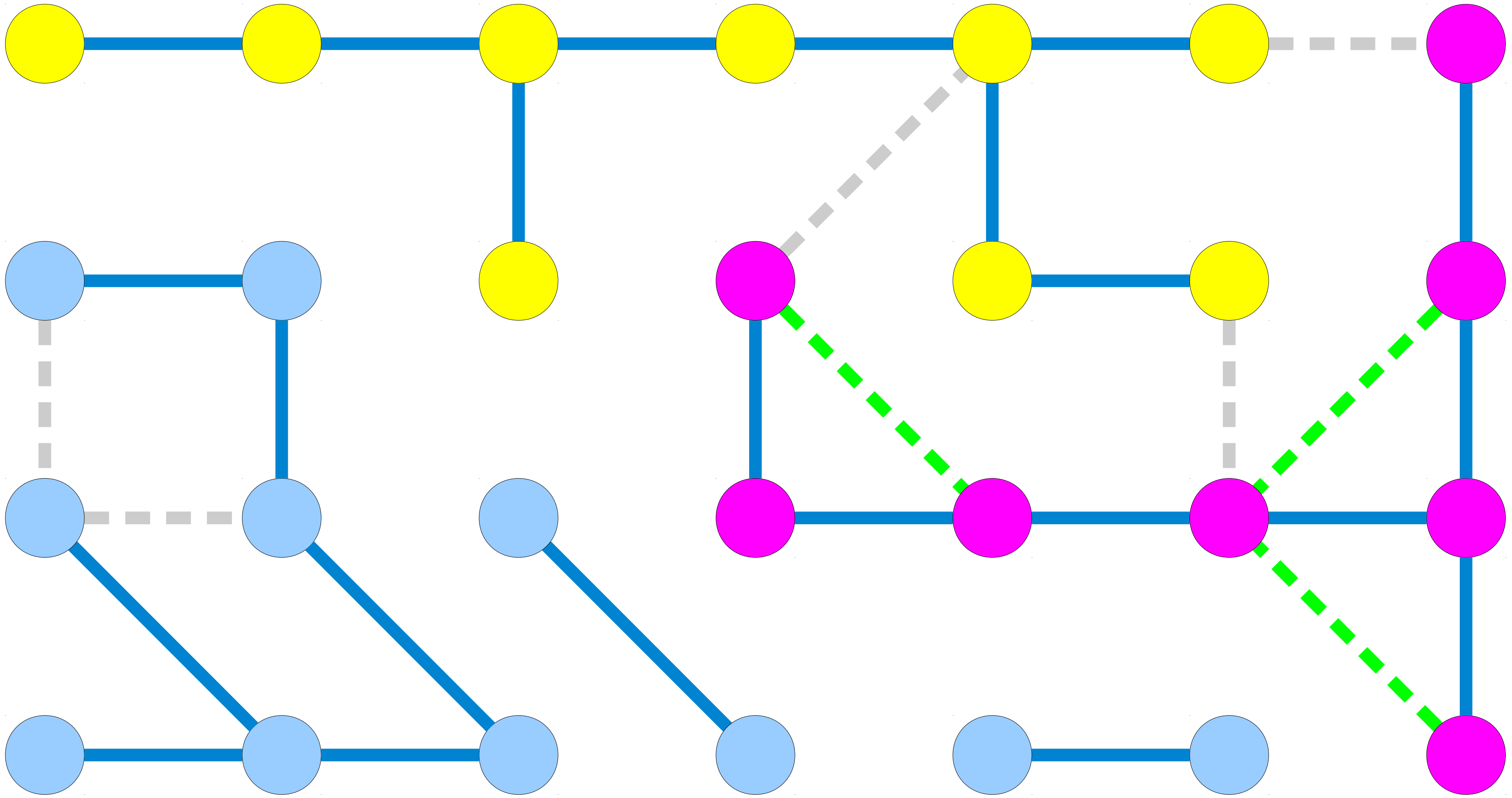


Every edge is either a **tree edge** in the forest $\mathscr{F}$ or an **auxiliary edge** running between two nodes in the same tree in $\mathscr{F}$.

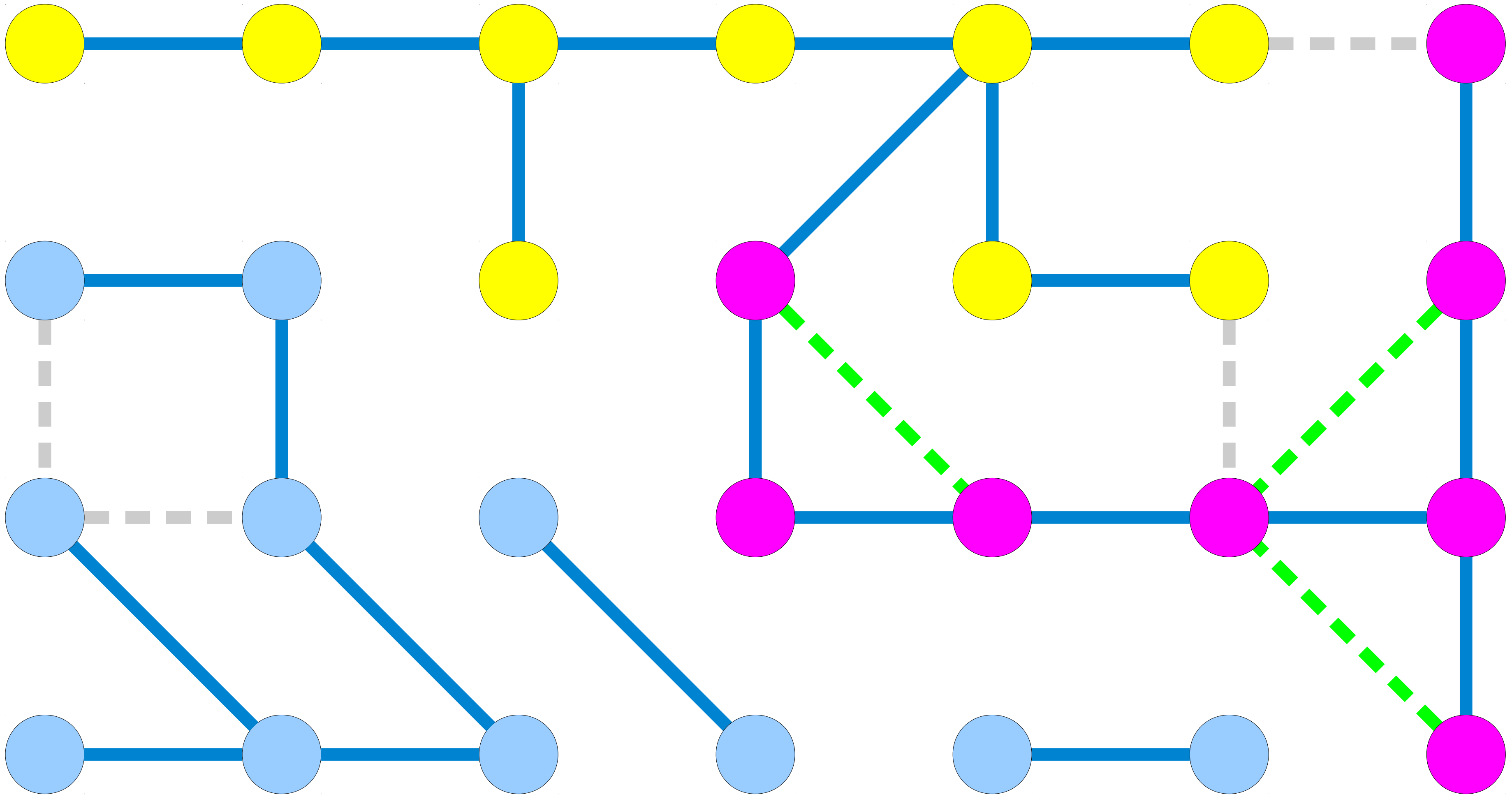
Maintaining a Forest



Maintaining a Forest



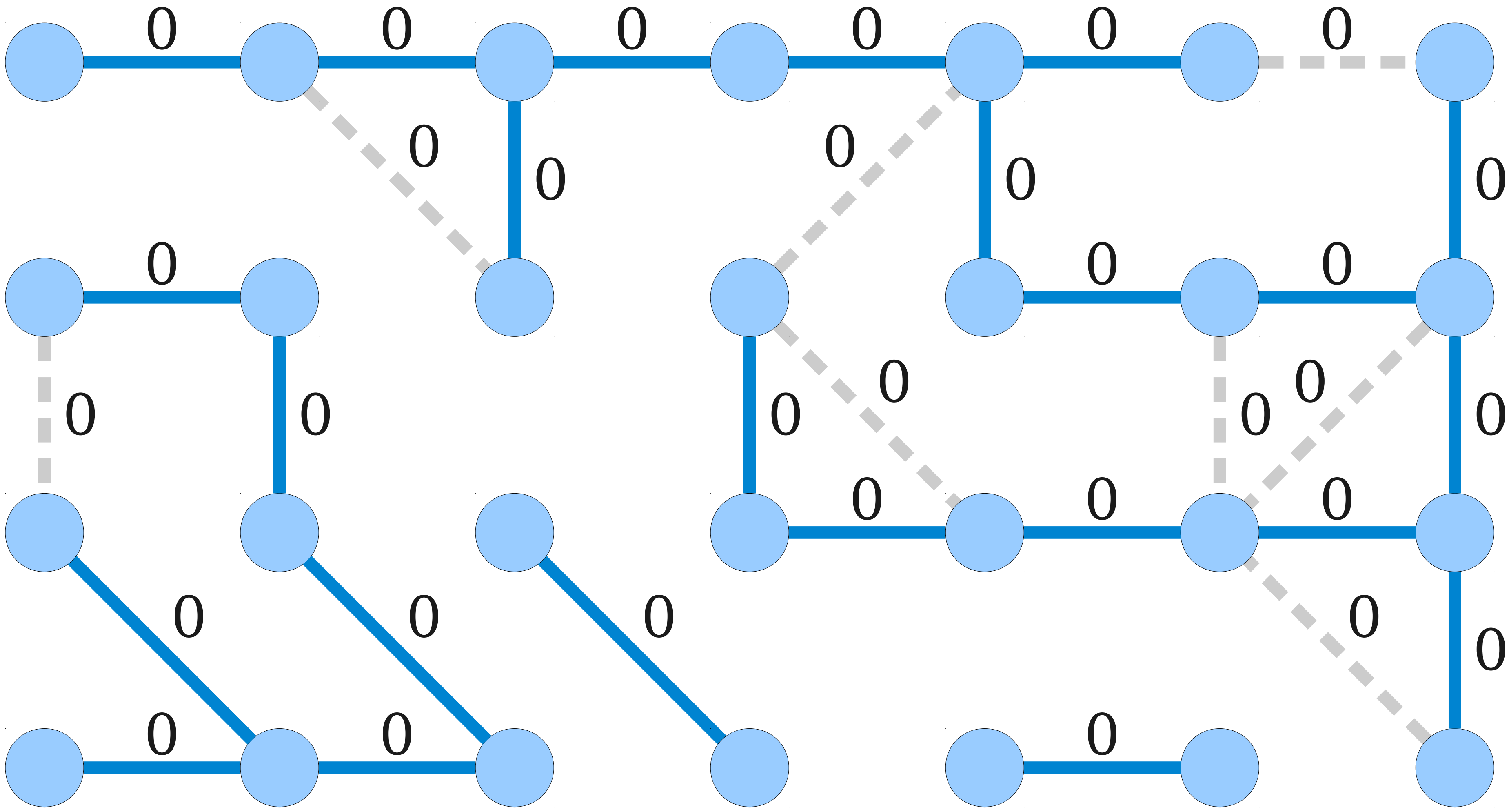
Maintaining a Forest



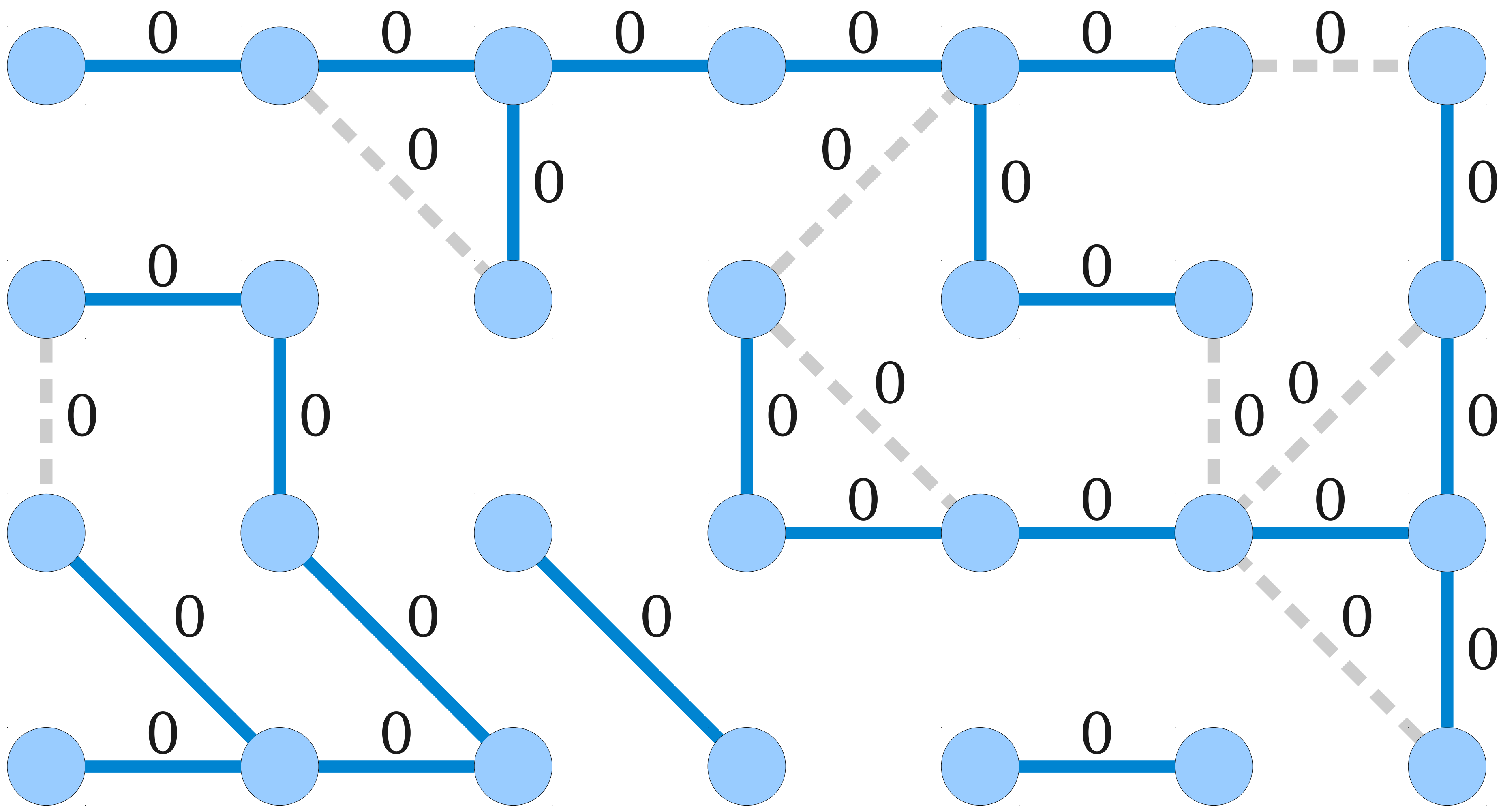
The Challenge

- **Goal:** Devise a way of storing edges such that we don't repeatedly rescan the same edges trying to glue trees together.
- **Idea:** Associate a “specificity” with each edge, initially 0.
- Edges with higher specificity refer to more restricted regions of the graph.
- Edges with lower specificity refer to more general regions of the graph.
- Adjust the specificity of edges in response to deletions.

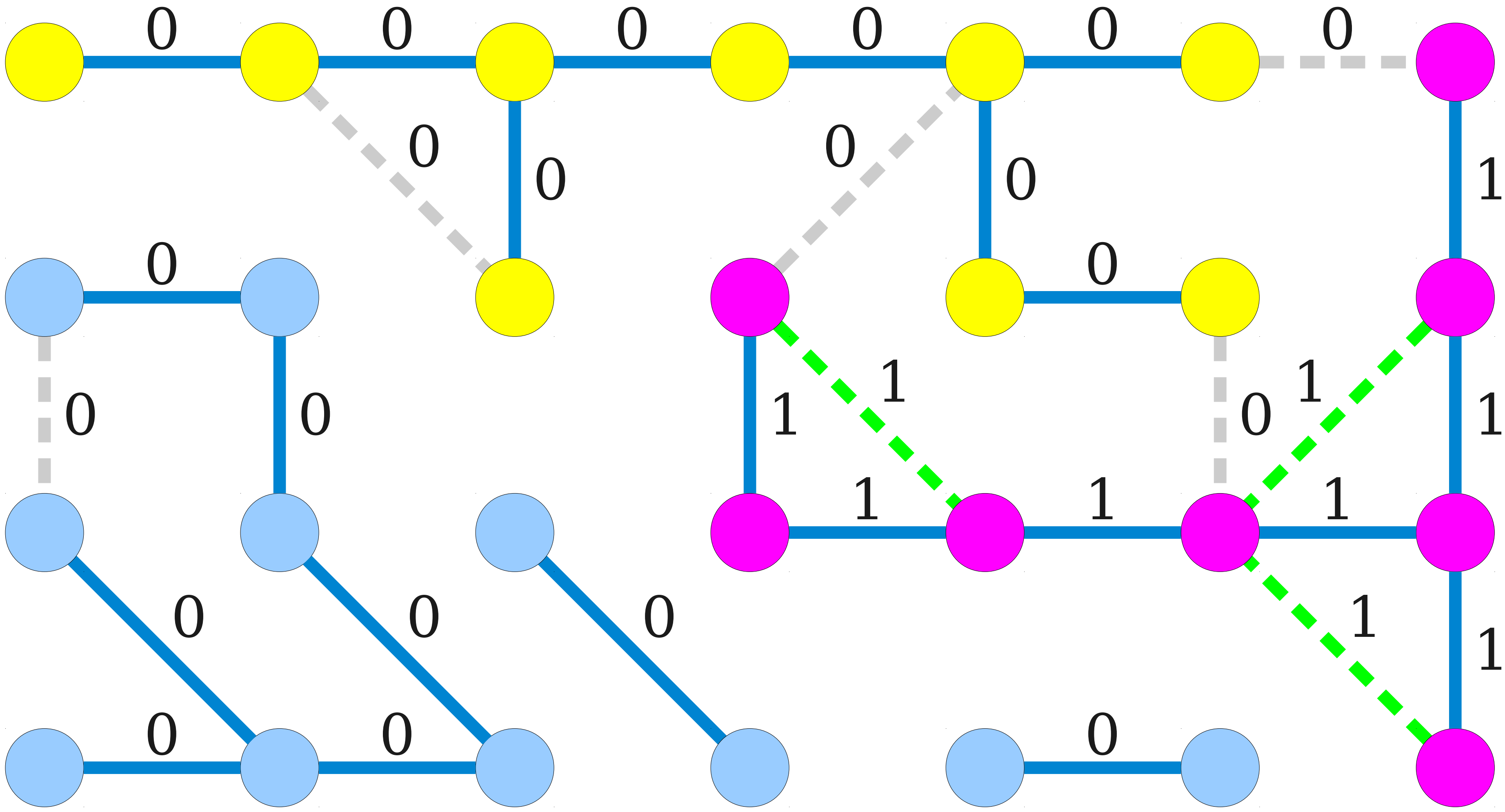
Edge Specificity



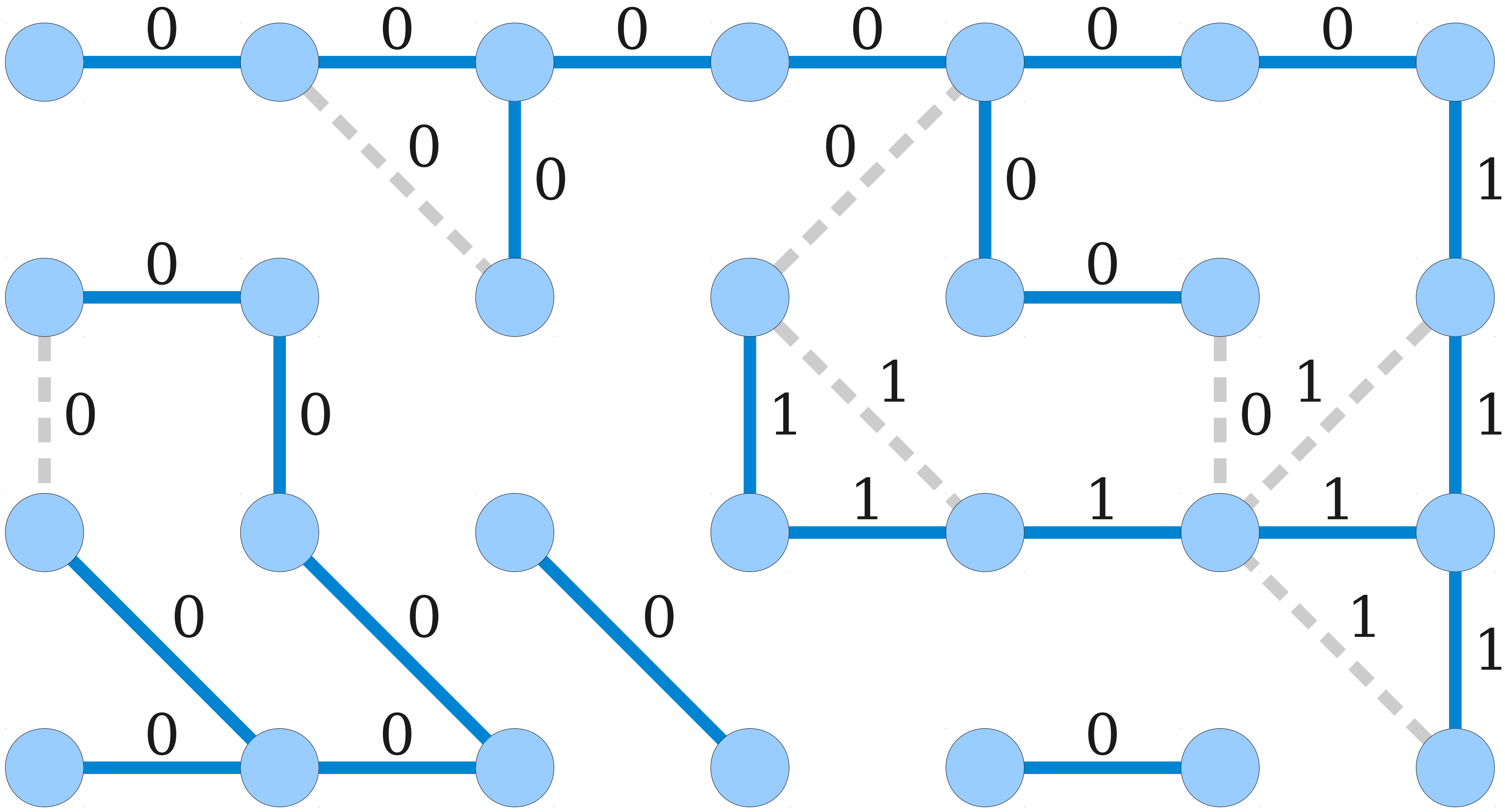
Edge Specificity



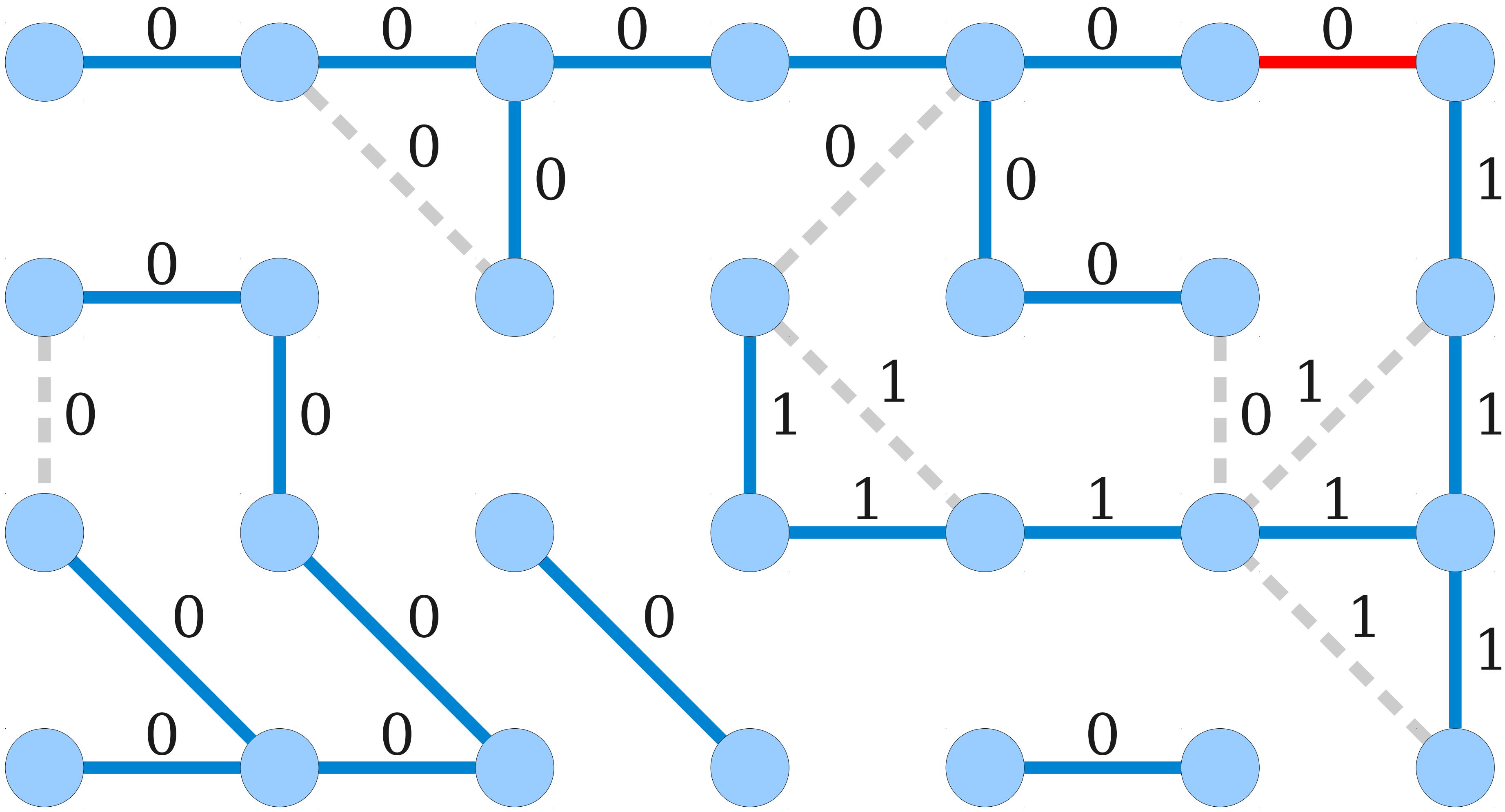
Edge Specificity



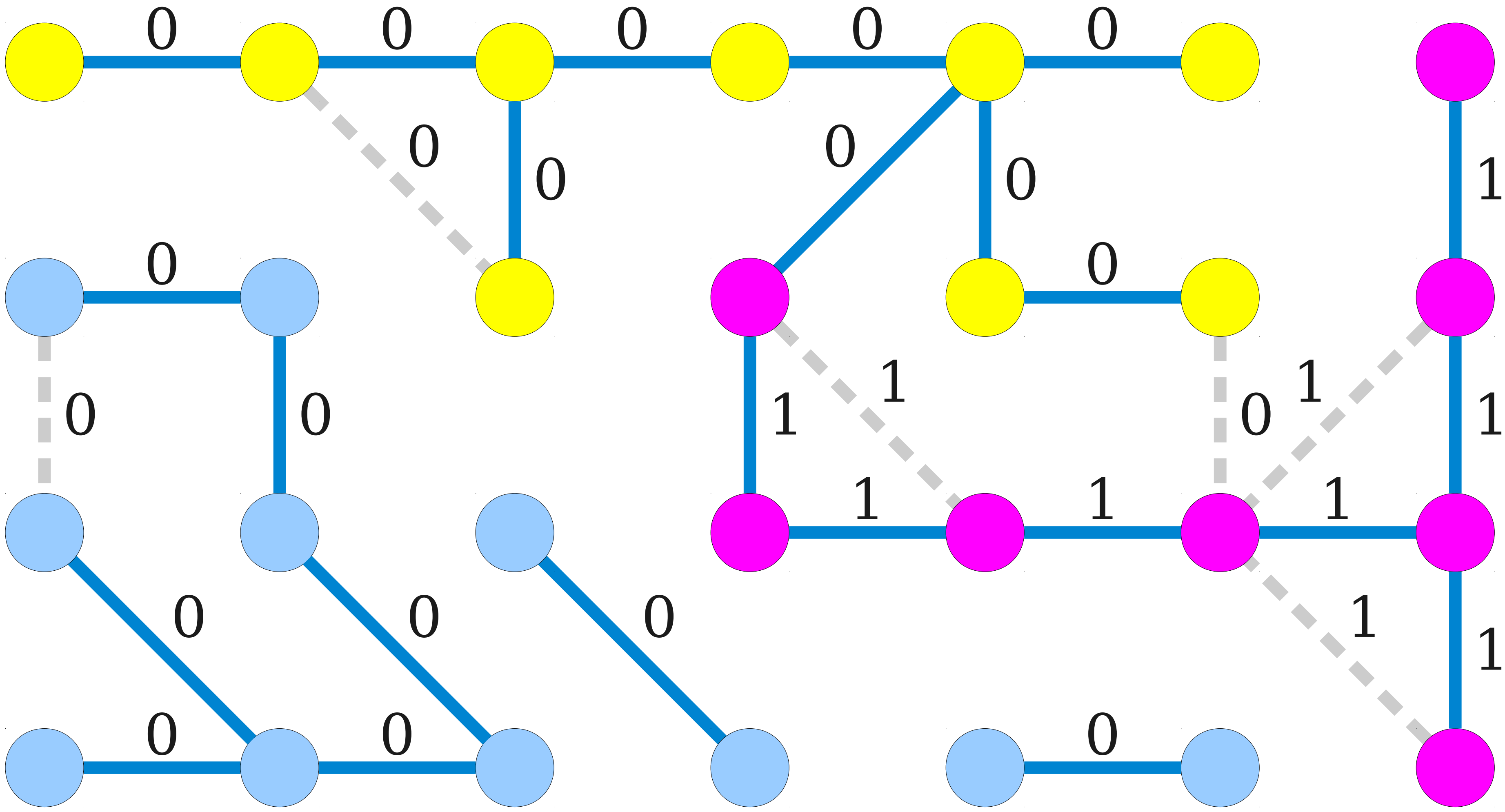
Edge Specificity



Edge Specificity



Edge Specificity



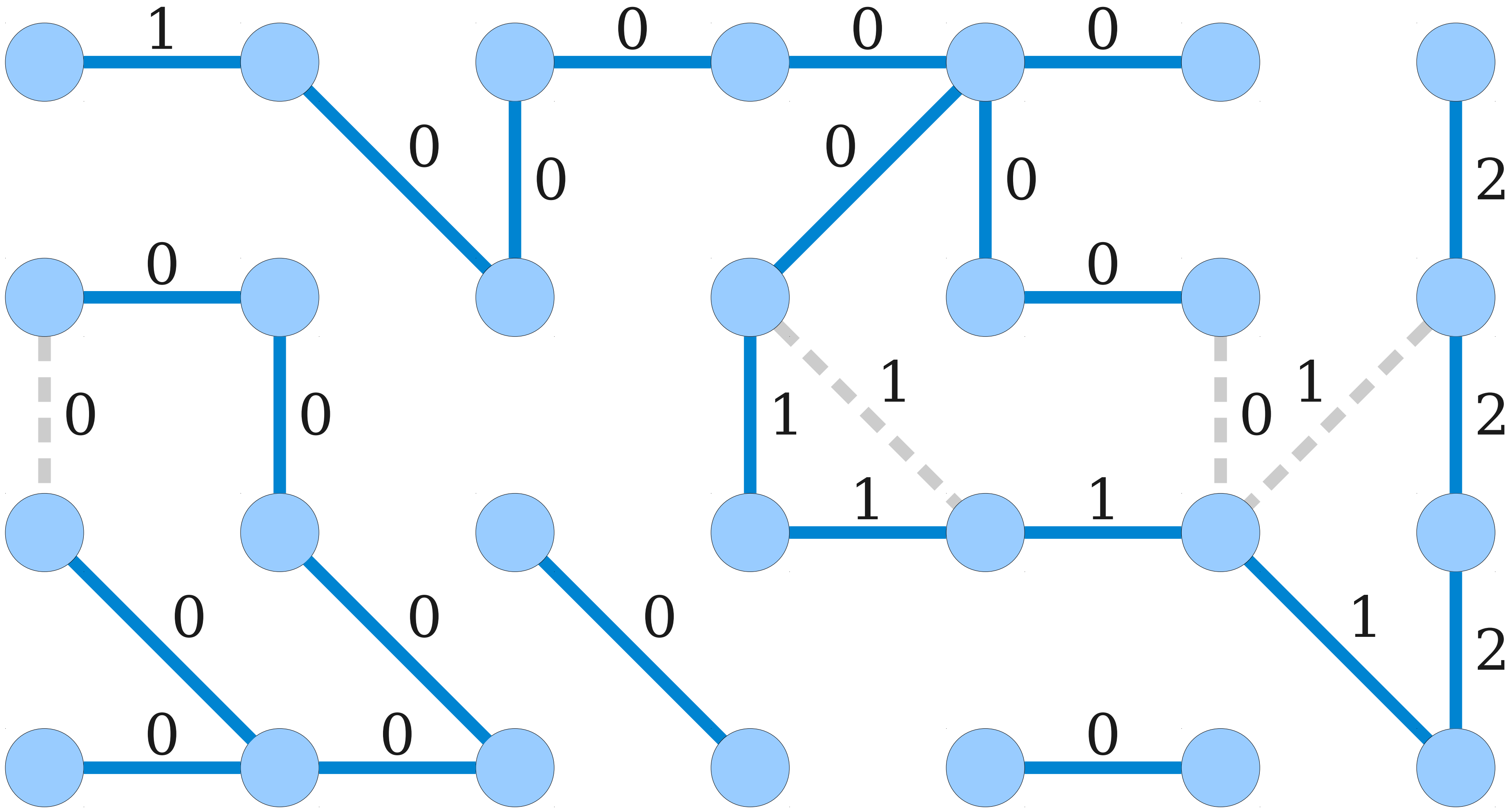
The Approach

- To delete a tree edge $\{u, v\}$ with specificity k:
  - Let T_u and T_v be the resulting trees.
  - Push all edges of specificity k in T_u up to specificity $k + 1$.
  - For all edges incident to T_u of specificity k:
    - If that edge connects T_u to T_v, add it to $\mathscr{F}$ and stop.
    - Otherwise, it connects T_u to itself, so increase its specificity.
  - If the previous iteration didn't reconnect T_u and T_v, repeat the above loop on specificity $k - 1$.

What Remains

- Some correctness concerns:
  - When searching for a replacement edge, we always search for edges with specificity no greater than the deleted edge.
  - How do we know that we won't “miss” a potential glue edge?
- The runtime analysis:
  - How efficient is this solution?

Arguing Correctness



The Invariant

- **Invariant 1:** $\mathscr{F}$ is a *maximum* spanning tree of G.
- Can quickly check that this holds in these cases:
  - Insertion: Add non-tree edge.
  - Insertion: Add tree edge.
  - Deletion: Delete non-tree edge.

The Invariant

- What happens if we delete a tree edge with specificity k?
  - Push up all edges in T_u with specificity k.
  - Search for edges of specificity k, k – 1, …, 0 to repair the tree.
- Things to check:
  - Does this preserve an MST?
  - Why can't we miss an edge?

The Runtime Analysis

The Representation

- Store a series of forests $\mathscr{F}_0$, $\mathscr{F}_1$, $\mathscr{F}_2$, …, .
- Forest $\mathscr{F}_k$ stores all edges at specificities k or greater.
  - Thus $\mathscr{F}_0 = \mathscr{F}$ and $\mathscr{F}_0 \supseteq \mathscr{F}_1 \supseteq \ldots$
- This enables us to query whether two nodes are connected by edges of level k or greater.

The Representation

- We can now think about our operations in terms of the hierarchical $\mathscr{F}_k$ forests.
- Inserting a new tree edge can be done in time O(log n) by ***link***ing the endpoints in the overall tree $\mathscr{F}_0$.
- Pushing a tree edge e of specificity k up to level $k + 1$ can be done in time O(log n) ***link***ing the endpoints of e in $\mathscr{F}_{k+1}$.
  - No need to ***cut*** them in $\mathscr{F}_k$; the forests are structure so that $\mathscr{F}_{k+1} \subseteq \mathscr{F}_k$.

Details We'll Ignore

- I'm going to gloss over some details, but you can trust me on these:
  - Auxiliary edges are stored in auxiliary data structures. Insertion or deletion takes time O(log n) each.
  - It's possible to iterate across all edges of level k incident to a given tree “efficiently.”
- Check the original paper for details; it's not really worth focusing on right now.

Runtime Analysis

- Connectivity queries can be answered in time O(log n) by querying $\mathscr{F}_0$.
- Inserting an edge takes time O(log n).
- Deleting an auxiliary edge takes time O(log n) (due to bookkeeping overhead.)

The Final Analysis

- Deleting a tree edge requires the following:
  - Deleting that tree edge from each of the forests in total time $O(r \log n)$, where r is the number of forests.
  - Possibly push up k edges from one layer to the next in total time $O(k \log n)$.
  - Possibly insert an edge into r forests in total time $O(r \log n)$.
- Total cost: $O(r \log n + k \log n)$
- This could be $O(m \log n)$ per operation!

A Problem

- Consider an n-clique.
- Delete some tree edge that splits the tree into a collection of one node and $n – 1$ nodes.
- If we push down the edges of the $(n – 1)$-node connected component, we could spend $\Theta(n^2 \log n)$ work trying to find an edge to connect it back to the remaining node.
- Repeat this on the $(n – 1)$-node tree in the forest, then repeat this on the $(n – 2)$-node tree, etc.
- Sums to $\Theta(n^3 \log n)$ total work across all n deletions.

Time-Out for Announcements!

Problem Set 7

- PS7 has been graded; we'll return it at the end of lecture today.

Presentations

- Final project presentations run all week.
- Presentations are open to everyone; feel free to stop on by to any of the talks (as long as it's not for your own data structure.)
  - They've been really, really great so far!

Your Questions!

“This class has been fantastic! If I like it, what other classes would you suggest taking?”

Awww, thanks! ☺

I'm planning on talking about this in depth in Wednesday's lecture. I hope that's okay with you!

“Splay trees seem like they would be useful in data compression (analogous to Huffman coding), but I can't find any common examples of their usage. Is there some example I'm missing, or is there a reason why they aren't as useful as expected?”

There has been some work done on this. Check “Applications of Splay Trees to Data Compression” by Douglas Jones as one source.

“Any idea what the curve will be like? What percentage of people do you see getting an A?”

I have no idea – it depends on final project grades. I usually put the median as the cutoff between B/B+, but depending on how people do I might be more generous.

“Recently we've developed bounds that don't reflect actual relative performance times. What is the role of this kind of work? Does it contribute to a larger theoretical understanding or is it mostly paper fodder and cool math?”

It's a little bit of both. ☺

We want to simultaneously build new tools and push back our understanding of theoretical limitations. The hard math enables us to do both.

“You're a really good teacher - what habits and techniques do you think set you apart? Besides practice, are there any specific things that made you better over time?”

Awww, thanks! ☺

I'd be happy to talk offline about this if you'd like. Most of it is just guesswork and learning from experience.

Back to CS166!

Where We Are

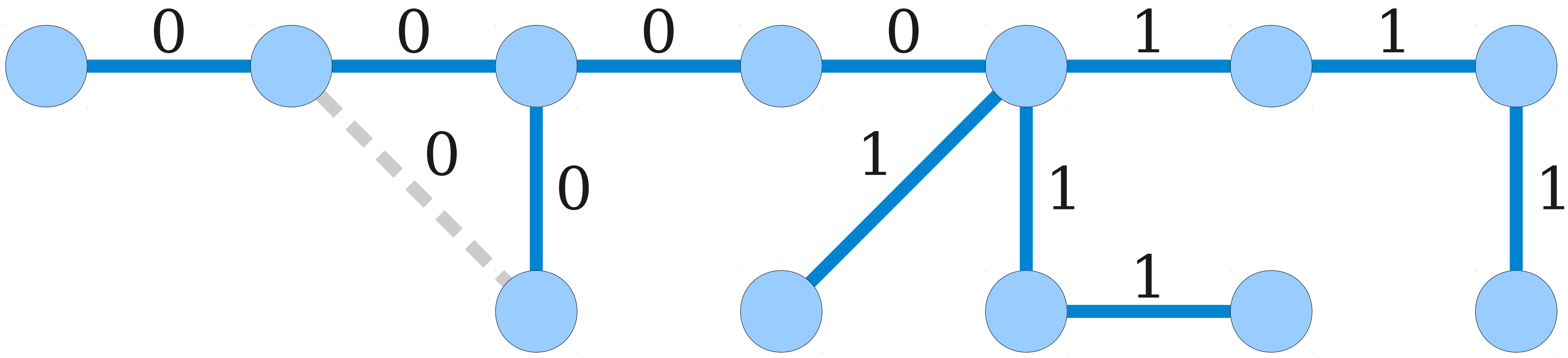
- We have a nifty data structure for maintaining dynamic connectivity.
- However, the time bounds on deletion are weak because we can push edges down many levels deep.
- Can we update our data structure so that deletions become less expensive?

Fixing the Problem

- On Problem Set Three, you saw a data structure for decremental connectivity.
- **Idea:** When splitting a tree in two, only relabel the nodes in the smaller connected component.
- This ensures each node is only relabeled at most O(log n) times and enables a clean amortized analysis.

Fixing the Problem

- **Modification:** When splitting a tree into two trees T_1 and T_2 by cutting an edge of specificity k, only increase the specificities of the level-k edges in the tree with fewer edges of level k or greater.



A Second Invariant

- **Recall:** We've hierarchically decomposed $\mathscr{F}$ into $\mathscr{F}_0$, $\mathscr{F}_1$, … where $\mathscr{F}_k$ consists of all tree edges of specificity k or greater.
- **Invariant 2:** For all k, the maximum number of nodes in any connected component of $\mathscr{F}_k$ is $n / 2^k$.
- **Corollary:** There are at most lg n distinct possible specificities.

Checking the Invariant

- Any CC in $\mathscr{F}_0 = \mathscr{F}$ has at most n nodes because there are n possible nodes.
- Insertions preserve the invariant because they only add edges to $\mathscr{F}_0$.
- Suppose we delete an edge in $\mathscr{F}_k$ and form two trees T_1 and T_2.
- We only increase the specificities of edges of level k in the smaller of T_1 and T_2.
- Since any CC in $\mathscr{F}_k$ has at most $n / 2^k$ nodes, this means any CC that gets pushed down can have at most $n / 2^{k+1}$ nodes.

The Final Analysis

- Deleting a tree edge requires the following:
  - Deleting that tree edge from each of the lg n forests in total time $O(\log^2 n)$.
  - Possibly push up k edges from one layer to the next in total time $O(k \log n)$.
  - Possibly insert an edge into lg n forests in total time $O(\log^2 n)$.
- Total cost: $O(\log^2 n + k \log n)$
- Each operation can still be pretty expensive.

The Final Analysis

- **Recall:** Edge levels range from 0 to lg *n*, and edge levels never decrease.
- **Idea:** Put lg *n* credits on each edge when that edge is added to the graph.
- Each credit can pay for the O(log *n*) work necessary to push an edge up a level.
- (This is the same idea from the decremental structure on the problem set.)

The Final Analysis

- Insertions have a base cost of O(log n).
- Adds lg n credits, each of which cost O(log n) units of work.
- Amortized cost: **O(log^2 n)**.

The Final Analysis

- Deletions have cost $O(\log^2 n + k \log n)$, where k is the number of edges promoted.
- Can spend one credit from each edge as it's promoted; won't run out of credits.
- Amortized cost: $\mathbf{O(\log^2 \boldsymbol{n})}$.

The Final Analysis

- The dynamic graph data structure supports the following operations in the indicated amortized runtimes:
  - ***is-connected***: $O(\log n)$
  - ***insert***: $O(\log^2 n)$
  - ***delete***: $O(\log^2 n)$
- This is significantly better than the naïve solution!
- Can we do better?

One Quick Speedup

- **Recall:** Each Euler tour tree is represented by a balanced BST.
- Lookup times in Euler tour trees is proportional to the tree height.
  - Walk from each node up to the root and compare whether the roots are the same.
- If we represent $\mathscr{F}_0$ (and just $\mathscr{F}_0$) using a B-tree of order Θ(log *n*), queries can be answered in time

$$O(\log_{\log n} n) = \mathbf{O(\log \boldsymbol{n} / \log \log \boldsymbol{n})}$$

The Final Analysis

- The dynamic graph data structure, with the B-tree modification, supports the following operations in the indicated amortized runtimes:
  - ***is-connected***: $O(\log n / \log \log n)$
  - ***insert***: $O(\log^2 n)$
  - ***delete***: $O(\log^2 n)$
- In practice, $\log n \leq 60$ and $\log \log n \leq 6$.

Going Forward

- Since this data structure was developed in 1999, there have been some new developments.
- If randomization is allowed, we can get these bounds:
  - ***is-connected***: $O(\log n / \log \log \log n)$
  - ***insert***: $O(\log n \cdot (\log \log n)^3)$ amortized
  - ***delete***: $O(\log n \cdot (\log \log n)^3)$ expected amortized
- A lower-bound of $\Omega(\log n)$ per insert or deletion is known to exist, and there's still a gap!

More Dynamic Problems

- Many other dynamic graph problems exist:
    - Maintaining an MST; can do in $O(\log^4 n)$ time per insertion or deletion.
    - Maintaining single-source or all-pairs shortest paths.
    - Maintaining reachability in a *directed* graph.
- All of these problems were solved in the static case 40+ years ago.
- We have somewhat decent solutions to the dynamic cases.
- ***This is an active area of research!***

Next Time

- **The Big Picture**
  - Wow, we covered a lot! What exactly did we see in this class?
- **Your Questions**
  - What didn't we cover that you wanted to learn in this class?
- **Where to Go From Here**
  - Next steps in theory (and in life?)