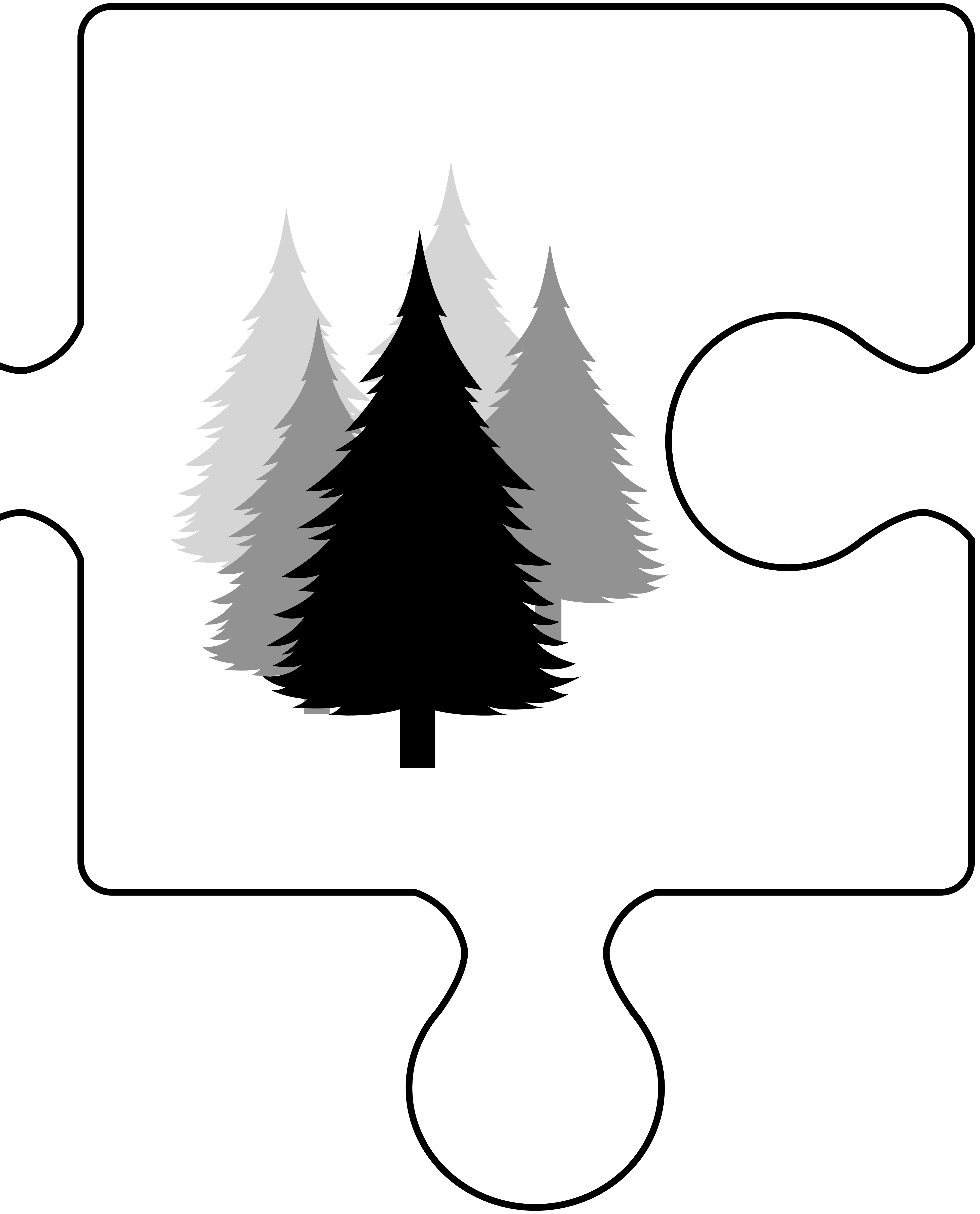
Loyola Marymount University

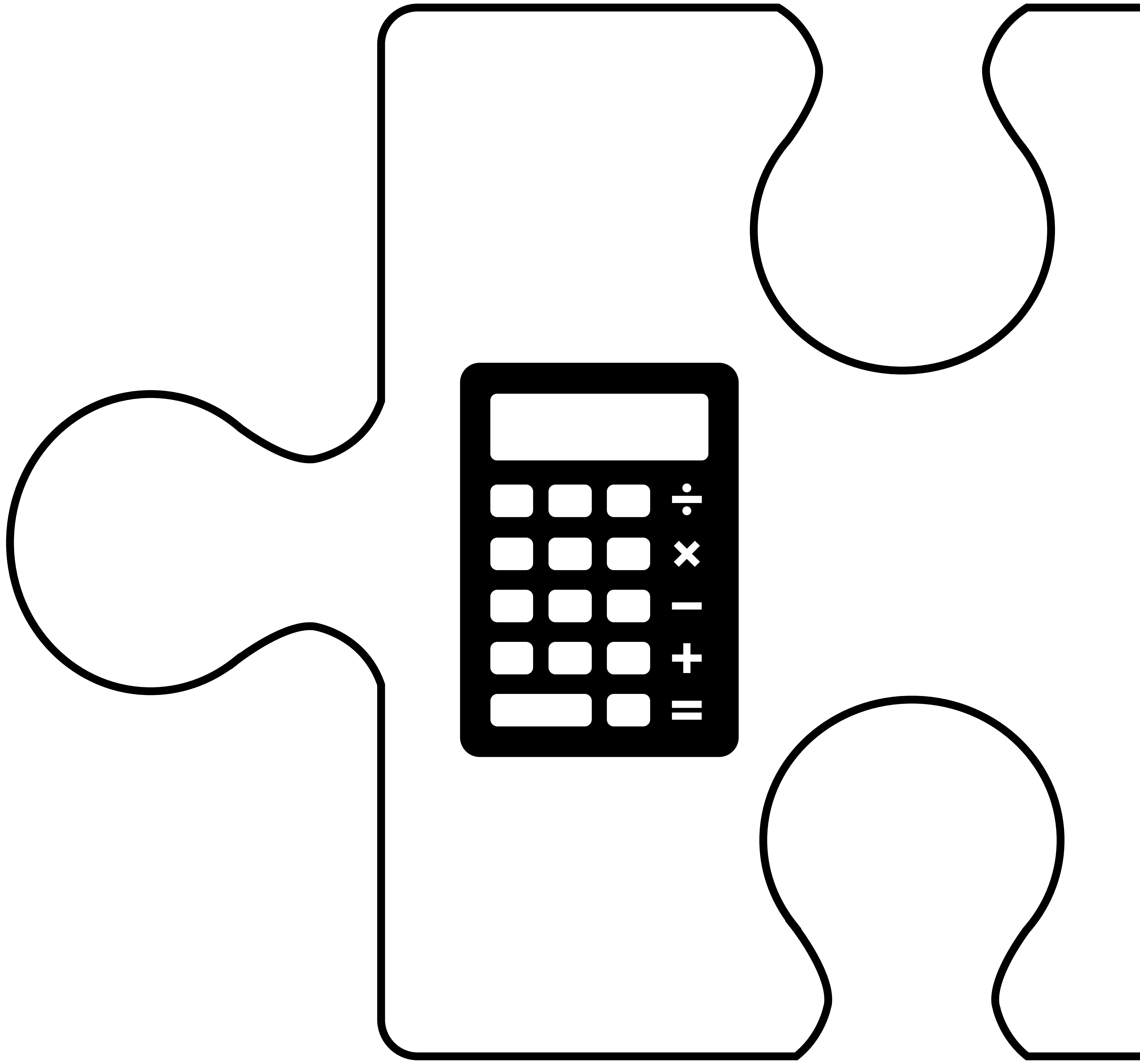
Loyola Marymount University
McNair Scholars Program

Tight Forests and the Chromatic Polynomial

Miliano Mikol

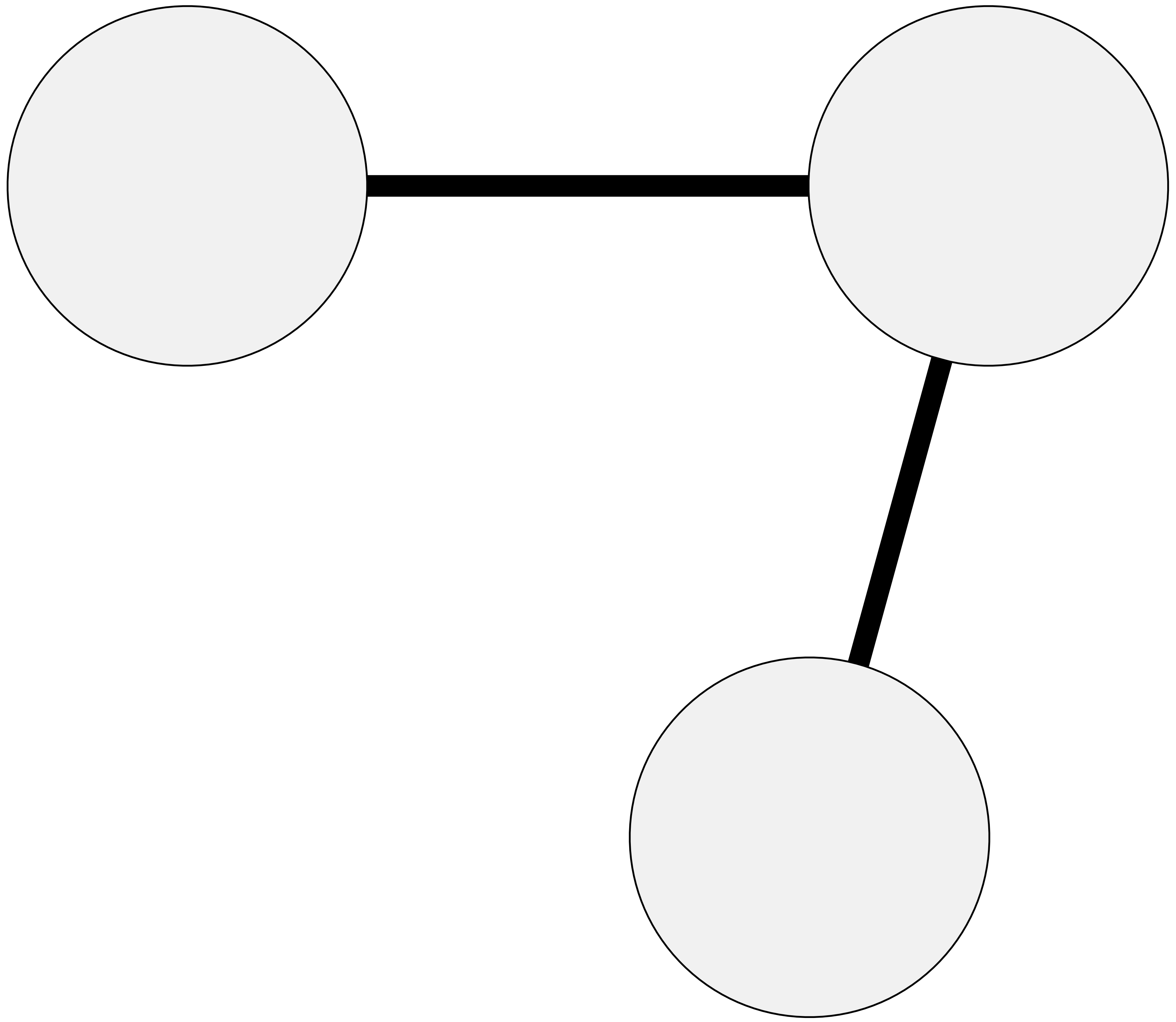
Research Adviser: Dr. Joshua Hallam, Ph.D, Mathematics

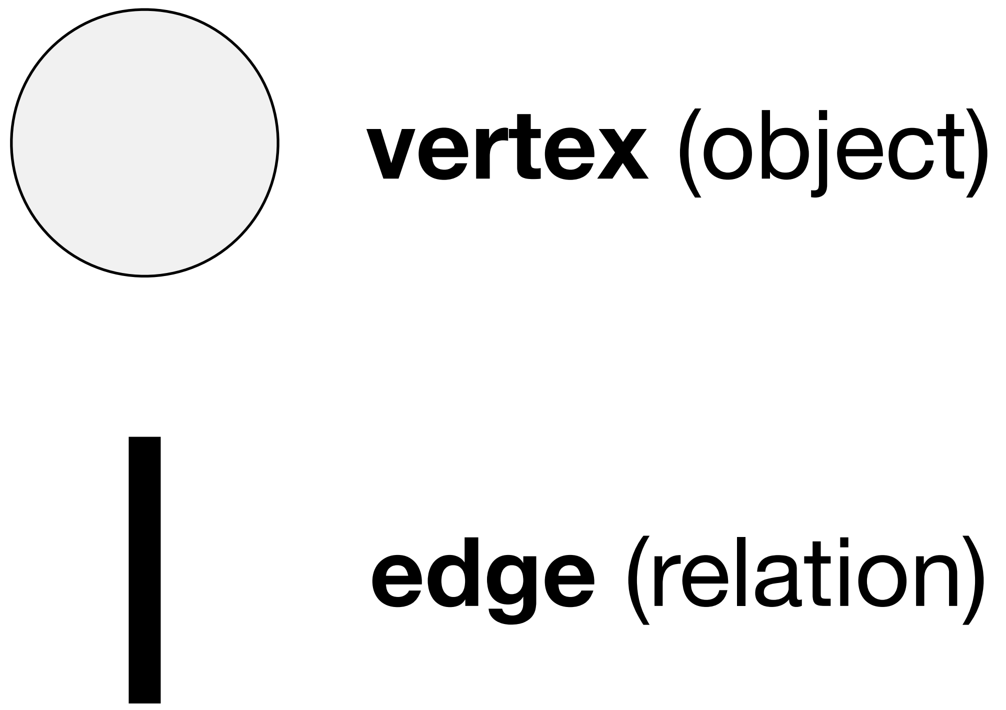




“Graph”

a structure that expresses relations





Why

graph theory is fundamental to many fields and technologies

Class Scheduling

Mathematics

Social Networks

Chemistry

Ecology Conservation

Google Maps Algorithms

DNA Sequencing

Medicine

Computer Science

Sociology

Search Algorithms

Network Security

Biology

Physics

File systems

Register Allocation

Engineering

Air Refueling Operations

You use graphs every day!

Suppose

this is our agenda today

Oil Change Appointment

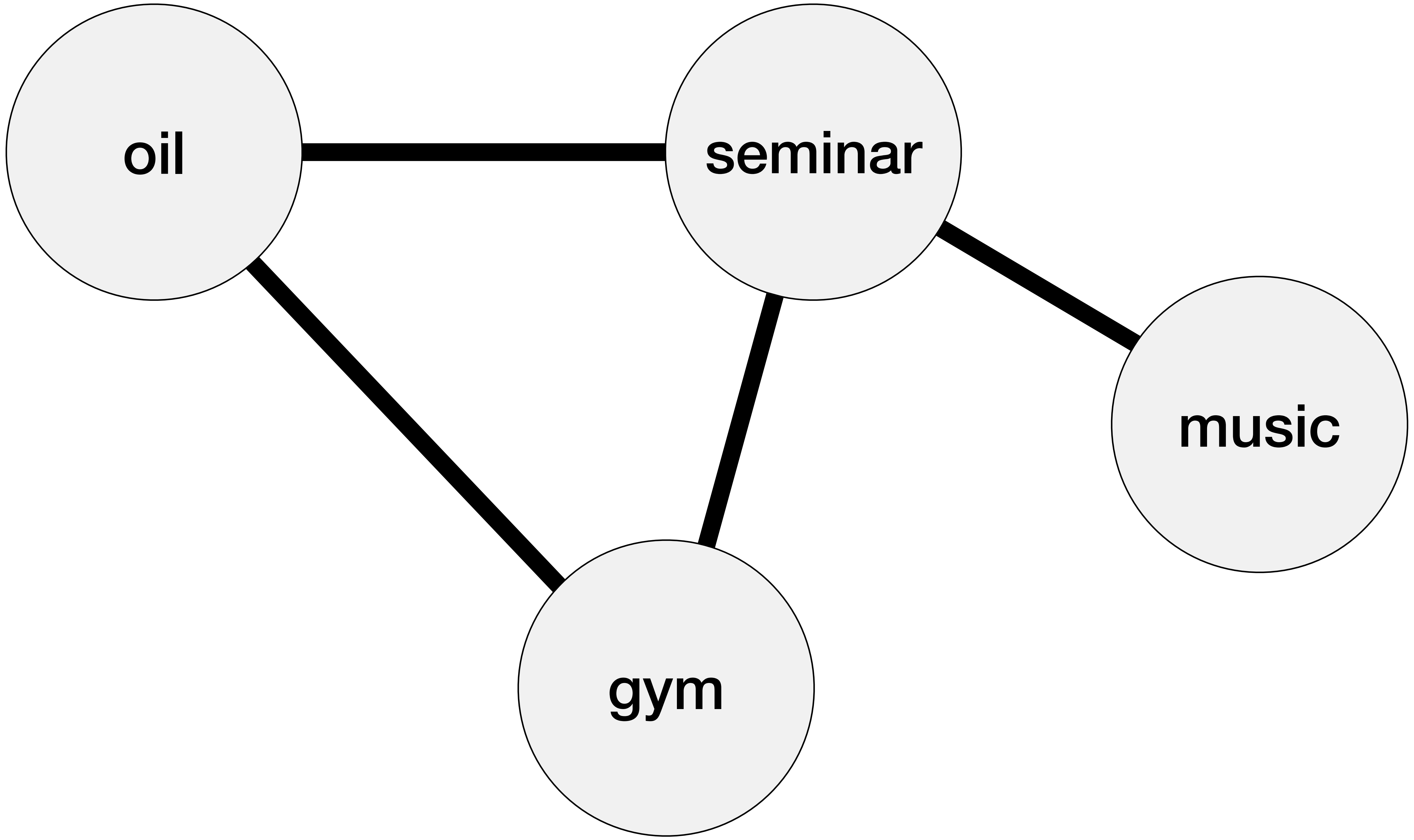
Attend Seminar

Listen To New Music

Hit The Gym

Goal

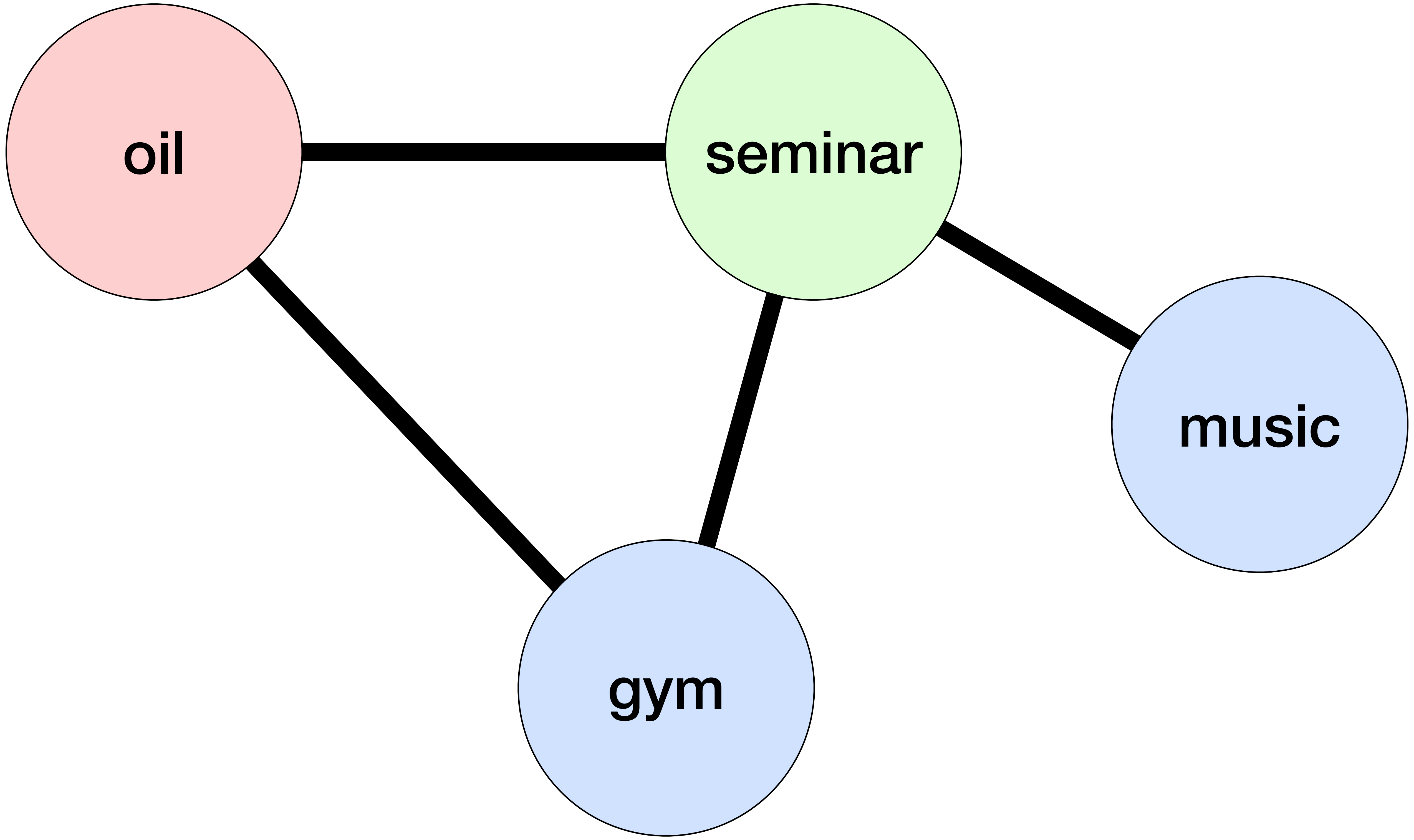
optimize our time



two tasks are connected **iff** they cannot be completed simultaneously

Solution

one out of twelve



Chromatic Polynomial

computes the number of colorings given t colors

$$\chi(G, t) = t(t-1)^2(t-2)$$

$$= 3(3-2)^2(3-2) \textbf{ [given } t = 3 \textbf{]}$$

$$= 12 \textbf{ possible ways to order the agenda}$$

The Project

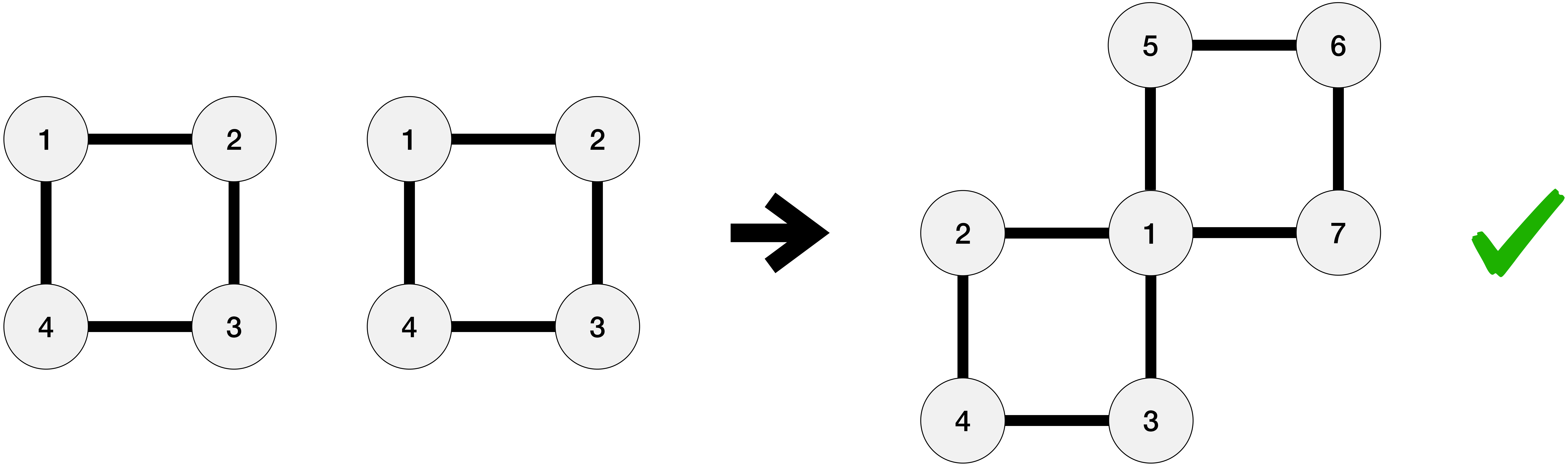
this equation is true for special graphs

$$TF(G,\ t) = (-1)^n\ \chi(G,\ -t)$$

…but which graphs are “special”?

Theorem Proven

gluing graphs in a particular way works



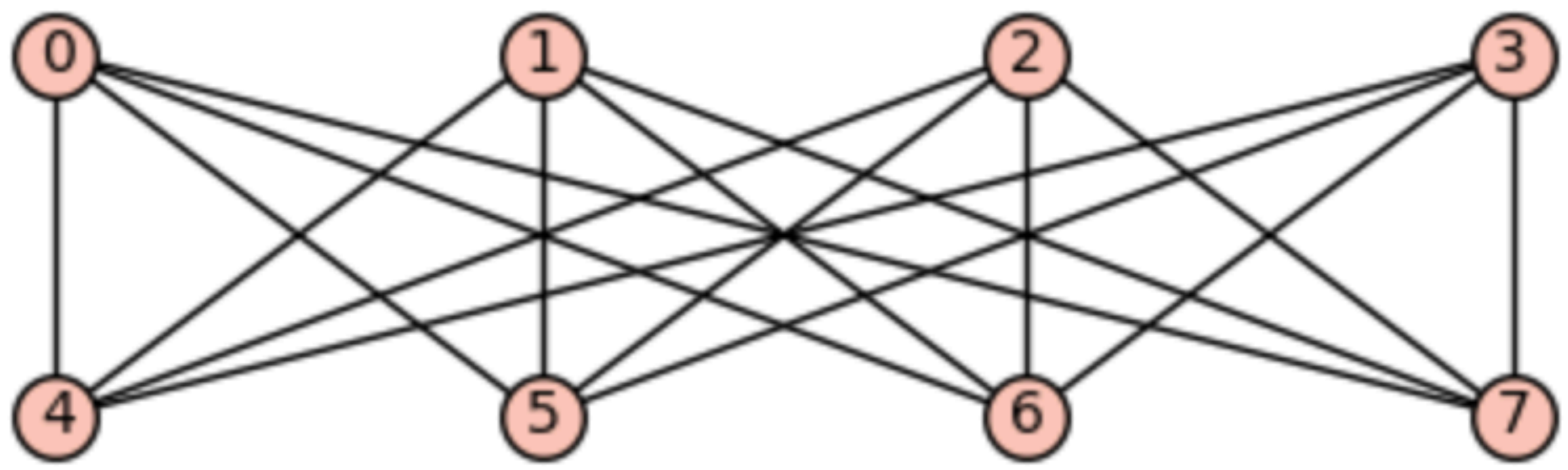
Step 1: Code

```python
def get_candidate_paths(G):
    candidate_paths = set([])

    def check_and_add(p):
        if (is_candidate_path(p)):
            candidate_paths.add(tuple(p))
        return

    for i in range(1, G.order()):
        for j in range(i + 1, G.order() + 1):
            for p in G.all_paths(i, j):
                check_and_add(p)
                p.reverse()
                check_and_add(p)

    return candidate_paths
```

```python
def is_candidate_path(P):
    min_len = 4

    if (len(P) < min_len):
        return False

    a, c = P[0], P[1]
    b, d = P[2], P[len(P) - 1]

    if a < b and b < c and P[len(P) - 1] < c:
        for i in range(min_len, len(P)):
            if (P[i - 1] < c):
                return False

        return True

    return False

def has_QPO(G, show_checks = False):
    candidate_paths = get_candidate_paths(G)
```

Step 2: Test/Optimize/Run

```
In [4]: G = graphs.CompleteBipartiteGraph(4, 4)
        deterministic_QPO_check(G)

start time: 2020-06-05 19:14:17.336624
checking this graph:
```




```
end time: 2020-06-05 19:32:02.029437
total time ran: 0:17:44.692813

Out[4]: 'has QPO: False'
```

Step 3: Prove

Theorem 3.5. *If n graphs with a quasi-perfect order are glued together such that for all G_i and G_{i+1}, G_i and G_{i+1} share the smallest vertex of G_{i+1}, then the resulting graph has a quasi-perfect order.*

Proof. We prove our result by induction.

(i) Base Case

($n = 1$) Trivial.

($n = 2$) Theorem 3.4.

Thus, the base case holds.

(ii) Inductive Step

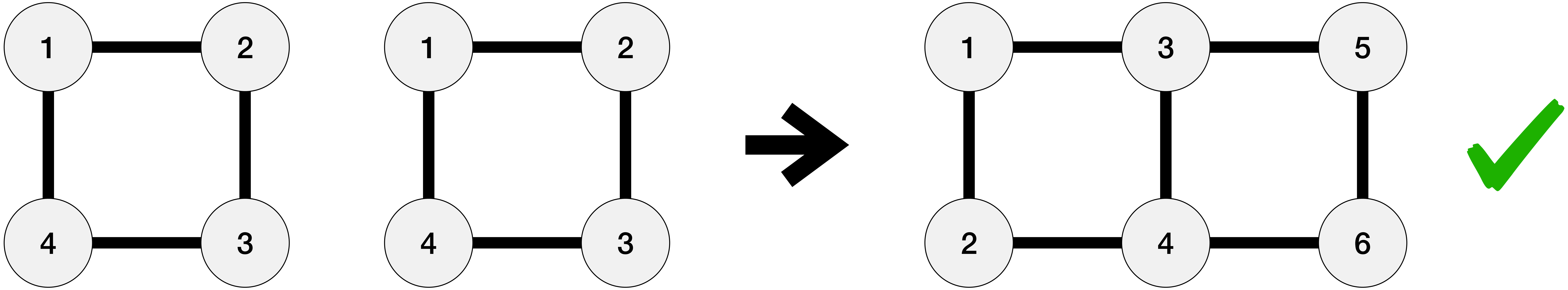
Inductive Hypothesis: Suppose that $1 \leq i \leq k$ graphs with a quasi-perfect order are glued together such that for all G_i and G_{i+1}, G_i and G_{i+1} share the smallest vertex of G_{i+1}, and the resulting graph has a quasi-perfect order.

We will show that if $k+1$ graphs with a quasi-perfect order are glued together such that for all G_i and G_{i+1}, G_i and G_{i+1} share the smallest vertex of G_{i+1}, then the resulting graph has a quasi-perfect order.

Let G be the graph formed by gluing k quasi-perfect order graphs such that, for all G_i and G_{i+1}, G_i and G_{i+1} share the smallest vertex of G_{i+1}. Let G_{k+1} be a graph with a quasi-perfect order. We now glue G and G_{k+1} such that, the two graphs share the smallest vertex of G_{k+1}. By the inductive hypothesis, G has a quasi-perfect order. Since both graphs have a quasi-perfect order and share the smallest vertex in G_{k+1}, the newly formed graph must have a quasi-perfect order by Theorem 3.4. Thus, our result holds by induction. $\square$

Theorem Proven

hope to prove for gluing in other different fashions



Thanks, Josh!
Thanks, McNair!