Problem Set 4

This fourth problem set explores set cardinality and graph theory. It serves as tour of the infinite (through set theory) and the finite (through graphs and their properties) and will give you a better sense for how discrete mathematical structures connect across these domains. Plus, you’ll get to see some pretty pictures and learn about why all this matters in the first place. ☺

Good luck, and have fun!

(No checkpoint!)
Due Friday, May 4th at 2:30PM
Problem One: Set Cardinalities

If \( A \) and \( B \) are sets, the **Cartesian product** of \( A \) and \( B \), denoted \( A \times B \), is the set
\[
\{ (x, y) \mid x \in A \land y \in B \}.
\]
Intuitively, \( A \times B \) is the set of all ordered pairs you can make by taking one element from \( A \) and one element from \( B \), in that order. For example, the set \( \{1, 2\} \times \{u, v, w\} \) is
\[
\{ (1, u), (1, v), (1, w), (2, u), (2, v), (2, w) \}.
\]

For the purposes of this problem, let’s let ★ and ☺ denote two arbitrary objects where ★ ≠ ☺. Over the course of this problem, we’re going to ask you to prove that \(|\mathbb{N} \times \{★, ☺\}| = |\mathbb{N}|\).

i. Draw a picture showing a way to pair off the elements of \( \mathbb{N} \times \{★, ☺\} \) with the elements of \( \mathbb{N} \) so that no elements of either set are uncovered or paired with multiple elements.

You might want to draw some pictures of the set \( \mathbb{N} \times \{★, ☺\} \) so that you can get a better visual intuition.

ii. Based on the picture you came up with in part (i), define a bijection \( f: \mathbb{N} \times \{★, ☺\} \rightarrow \mathbb{N} \). The inputs to this function will be elements of \( \mathbb{N} \times \{★, ☺\} \), so you can define your function by writing
\[
f(n, x) = \text{___________________________}
\]
where \( n \in \mathbb{N} \) and \( x \in \{★, ☺\} \).

In defining this function, you cannot assume ★ or ☺ are numbers, since they’re arbitrary values out of your control. See if you can find a way to define this function that doesn’t treat ★ and ☺ algebraically.

iii. Prove that the function you came up with in part (ii) is a bijection.

The result you’ve proved here essentially shows that \( 2\aleph_0 = \aleph_0 \). Isn’t infinity weird?

Problem Two: Understanding Diagonalization

Proofs by diagonalization are tricky and rely on nuanced arguments. In this problem, we’ll ask you to review the formal proof of Cantor’s theorem to help you better understand how it works. (Please read the Guide to Cantor's Theorem before attempting this problem.)

i. Consider the function \( f: \mathbb{N} \rightarrow \wp(\mathbb{N}) \) defined as \( f(n) = \emptyset \). Trace through our formal proof of Cantor’s theorem with this choice of \( f \) in mind. In the middle of the argument, the proof defines some set \( D \) in terms of \( f \). Given that \( f(n) = \emptyset \), what is that set \( D \)? Provide your answer without using set-builder notation. Is it clear why \( f(n) \neq D \) for any \( n \in \mathbb{N} \)?

Make sure you can determine what the set \( D \) is both by using the visual intuition behind Cantor’s theorem and by symbolically manipulating the formal definition of \( D \) given in the proof.

ii. Let \( f \) be the function from part (i). Find a set \( S \subseteq \mathbb{N} \) such that \( S \neq D \), but \( f(n) \neq S \) for any \( n \in \mathbb{N} \).

Justify your answer. This shows that while the diagonalization proof will always find some set \( D \) that isn’t covered by \( f \), it won’t find every set with this property.

iii. Repeat part (i) of this problem using the function \( f: \mathbb{N} \rightarrow \wp(\mathbb{N}) \) defined as
\[
f(n) = \{ m \in \mathbb{N} \mid m \geq n \}
\]
Now what do you get for the set \( D \)? Is it clear why \( f(n) \neq D \) for any \( n \in \mathbb{N} \)?

iv. Repeat part (ii) of this problem using the function \( f \) from part (iii).
Problem Three: Simplifying Cantor's Theorem?

In our proof of Cantor's theorem, we proved that $|S| \neq |\wp(S)|$ by using a diagonal argument. Below is a purported proof that $|S| \neq |\wp(S)|$ that doesn't use a diagonal argument:

**Theorem:** If $S$ is a set, then $|S| \neq |\wp(S)|$.

**Proof:** Let $S$ be any set and consider the function $f: S \to \wp(S)$ defined as $f(x) = \{x\}$. To see that this is a valid function from $S$ to $\wp(S)$, note that for any $x \in S$, we have $\{x\} \subseteq S$. Therefore, $\{x\} \in \wp(S)$ for any $x \in S$, so $f$ is a legal function from $S$ to $\wp(S)$.

Let's now prove that $f$ is injective. Consider any $x_1, x_2 \in S$ where $f(x_1) = f(x_2)$. We'll prove that $x_1 = x_2$. Because $f(x_1) = f(x_2)$, we have $\{x_1\} = \{x_2\}$. Since two sets are equal if and only if their elements are the same, this means that $x_1 = x_2$, as required.

However, $f$ is not surjective. Notice that $\emptyset \in \wp(S)$, since $\emptyset \subseteq S$ for any set $S$, but that there is no $x$ such that $f(x) = \emptyset$; this is because $\emptyset$ contains no elements and $f(x)$ always contains one element. Since $f$ is not surjective, it is not a bijection. Thus $|S| \neq |\wp(S)|$. ■

Unfortunately, this argument is incorrect. What's wrong with this proof? Justify your answer by pointing to a specific claim that's made here that’s incorrect and explaining why it’s incorrect.

Problem Four: Paradoxical Sets

What happens if we take *absolutely everything* and throw it into a set? If we do, we would get a set called the **universal set**, which we denote $\mathcal{U}$:

$$\mathcal{U} = \{ x \mid x \text{ exists } \}$$

Absolutely everything would belong to this set:

$$1 \in \mathcal{U} \quad \mathbb{N} \in \mathcal{U} \quad \text{CS103} \in \mathcal{U} \quad \text{whimsy} \in \mathcal{U}$$

In fact, we'd even have $\mathcal{U} \in \mathcal{U}$, which is strange but not immediately a problem. Unfortunately, the set $\mathcal{U}$ doesn't actually exist, as its existence would break mathematics.

i. Prove that if $A$ and $B$ are arbitrary sets where $A \subseteq B$, then $|A| \leq |B|$.

**Look at the online Guide to Cantor’s Theorem. Formally speaking, if you want to prove that $|A| \leq |B|$, what do you need to prove? Your answer should involve defining some sort of function between $A$ and $B$ and proving that function has some specific property or properties.**

ii. Using your result from (i), prove that if $\mathcal{U}$ exists, then $|\wp(\mathcal{U})| \leq |\mathcal{U}|$.

The **Cantor-Bernstein-Schroeder Theorem** says that if $A$ and $B$ are sets such that $|A| \leq |B|$ and $|B| \leq |A|$, then $|A| = |B|$. We can use this to show that if $A$ and $B$ are sets where $|A| \leq |B|$, then $|B| \not< |A|$. You may use this fact in part iii.

iii. Using your results from part ii of this problem and the above note, prove that $\mathcal{U}$ does not exist.

The result you've proven shows that there is a collection of objects (the collection of everything that exists) that cannot be put into a set. When this was discovered at the start of the twentieth century, it led to a reexamination of logical reasoning itself and a more formal definition of what objects can and cannot be gathered into a set. If you're curious to learn more about what came out of that, take Math 161 (Set Theory) or Phil 159 (Non-Classical Logic).
Problem Five: Independent Sets

An independent set in a graph $G = (V, E)$ is a set $I \subseteq V$ with the following property:

$$\forall u \in I. \forall v \in I. \{u, v\} \notin E.$$ 

This question explores independent sets and their properties.

i. Explain what an independent set is in plain English and without making reference to first-order logic. No justification is necessary.

You may want to draw some pictures of graphs to see what independent sets look like. Don’t just come up with a literal translation of the first-order logic formula above; see if you can find a simple explanation of what independent sets are.

ii. Download the starter files for Problem Set Four from the website, then open the file GraphTheory.cpp and implement a function

```cpp
bool isIndependentSet(Graph G, std::set<Node> I)
```

that takes as input a graph $G$ and a set $I$, then returns whether $I$ is an independent set in $G$. The definition of the Graph type is provided in GraphTheory.h.

Our provided starter code contains logic that, given a graph $G$, finds the largest independent set in $G$ by making a lot of repeated calls to isIndependentSet. You might want to look over some of the sample graphs to get a feel for what large independent sets look like.

The size of an independent set is the number of nodes in it. Formally speaking, if $I$ is an independent set, then the size of $I$ is given by $|I|$. The independence number of a graph $G$, denoted $\alpha(G)$, is the size of the largest independent set in $G$. (Note that there can be many different independent sets in a graph $G$ that are all tied for the largest.)

iii. Edit the file PartA.graph in the res/ directory to define a graph $G$ where $G$ has exactly five nodes and $\alpha(G) = 5$.

iv. Edit the file PartB.graph in the res/ directory to define a graph $G$ where $G$ has exactly five nodes and $\alpha(G) = 1$.

A graph can contain multiple different independent sets.

vi. Prove or disprove: if $G = (V, E)$ is a graph and $I_1$ and $I_2$ are independent sets in $G$, then $I_1 \cap I_2$ is an independent set in $G$.

Independent sets are specified using a definition given in first-order logic. Make sure your proof is structured around that definition, the same way that proofs of reflexivity, symmetry, etc. are structured around those first-order definitions.

Problem Six: Vertex Covers

A vertex cover in a graph $G = (V, E)$ is a set $C \subseteq V$ with the following property:

$$\forall u \in V. \forall v \in V. \{u, v\} \in E \rightarrow u \in C \lor v \in C.$$ 

This question explores vertex covers and their properties.

i. Translate the definition of a vertex cover into plain English. No justification is necessary.

As before, you may want to draw pictures. See if you can find an explanation that’s more than just a literal translation of the above statement.
ii. Implement a function

```cpp
bool isVertexCover(Graph G, std::set<Node> C)
```

that takes as input a graph \( G \) and a set \( C \), then returns whether \( C \) is a vertex cover of \( G \).

Our provided starter code contains logic that, given a graph \( G \), finds the smallest vertex cover in \( G \) by making a lot of repeated calls to `isVertexCover`. You might want to explore some of the sample graphs to get a feel for what vertex covers look like.

The size of a vertex cover is the number of nodes in it. Formally speaking, if \( C \) is a vertex cover, then the size of \( C \) is given by \(|C|\). The **vertex cover number** of a graph \( G \), denoted \( \tau(G) \), is the size of the smallest vertex cover of \( G \). (Note that there can be many different vertex covers in a graph \( G \) that are all tied for the smallest.)

iii. Edit the file `PartC.graph` in the `res/` directory to define a graph \( G \) with exactly five nodes where \( \tau(G) = 0 \).

iv. Edit the file `PartD.graph` in the `res/` directory to define a graph \( G \) with exactly five nodes where \( \tau(G) = 4 \).

As with independent sets, graphs can contain multiple different vertex covers.

vi. Prove or disprove: if \( G = (V, E) \) is a graph and \( C_1 \) and \( C_2 \) are vertex covers of \( G \), then \( C_1 \cap C_2 \) is a vertex cover of \( G \).

Vertex covers have some really cool applications. Check out this Numberphile video for one of them!

**Problem Seven: Chromatic and Clique Numbers**

Recall from lecture that a **\( k \)-vertex coloring** of a graph is a way of coloring each node in the graph one of up to \( k \) different colors such that no two adjacent nodes are the same color. The **chromatic number** of a graph, denoted \( \chi(G) \), is the minimum value of \( k \) for which a \( k \)-vertex coloring exists.

i. Implement a function

```cpp
bool isKVertexColoring(Graph G, 
                        std::map<Node, Color> colors,
                        std::size_t k);
```

that takes as input a graph, a mapping from nodes in the graph to colors, and a number \( k \), then returns whether the indicated coloring is a \( k \)-vertex coloring. You can assume that the map has one key for each node in the graph and that the only keys in the map are nodes in \( G \). (The type `std::size_t` represents a natural number.)

Our provided starter code contains some logic that, given a graph \( G \), finds a minimum \( k \)-vertex-coloring of \( G \) by making a lot of calls to your `isKVertexColoring` function. We recommend taking some time to look at a few sample graphs and their minimum colorings – they’re quite pretty!

Here’s a new definition. A **clique** in a graph \( G = (V, E) \) is a set \( K \subseteq V \) with the following property:

\[
\forall u \in K. \forall v \in K. (u \neq v \rightarrow \{u, v\} \in E).
\]

This question explores the connection between cliques and chromatic numbers.

ii. Translate the definition of a clique into plain English. No justification is necessary.

iii. Implement a function

```cpp
bool isClique(Graph G, std::set<Node> K)
```
that takes as input a graph $G$ and a set $K$, then returns whether $K$ is a clique in $G$.

Our provided starter code contains some logic that, given a graph $G$, finds the largest clique in $G$ by making a lot of calls to your `isclique` function. You may want to take a look at some of the provided sample graphs to see what large cliques look like.

The size of a clique $K$, denoted $|K|$, is the number of nodes in $K$. The **clique number** of a graph, denoted $\omega(G)$, is the size of the largest clique in $G$. (Note that there can be many different cliques in a graph $G$ that are all tied for the largest.)

iv. Prove that if $G$ is a graph, then $\chi(G) \geq \omega(G)$.

*We’re expecting you to write a formal proof here. It may be easiest to do this by contradiction.*

v. Edit the file `PartE.graph` in the `res/` directory to contain a graph $G$ where $\chi(G) \neq \omega(G)$.

This shows that, in general, the chromatic and clique numbers of a graph don’t have to be equal.

*Aim to find the smallest example that you can. Although you aren’t required to submit the simplest example possible and we aren’t asking for an explanation as to why your answer is correct, you should not feel satisfied with your answer unless you can justify why it’s got to be the simplest answer possible.*

### Problem Eight: Chromatic and Independence Numbers

In Problem Seven, you explored the connection between clique numbers and chromatic numbers. This problem explores the connection between independence numbers and chromatic numbers.

Let $n$ be an arbitrary positive natural number. Prove that if $G$ is an arbitrary undirected graph with exactly $n$ nodes, then the product $\chi(G) \alpha(G) \geq n$.

### Problem Nine: Bipartite Graphs

An undirected graph $G = (V, E)$ is called a **bipartite graph** if there exist two sets $V_1$ and $V_2$ such that

- every node $v \in V$ belongs to exactly one of $V_1$ and $V_2$, and
- every edge $e \in E$ has one endpoint in $V_1$ and the other in $V_2$.

The sets $V_1$ and $V_2$ here are called **bipartite classes** of $G$. To help you get a better intuition for bipartite graphs, suppose that you have a group of people and a list of restaurants. You can illustrate which people like which restaurants by constructing a bipartite graph where $V_1$ is the set of people, $V_2$ is the set of restaurants, and there's an edge from a person $p$ to a restaurant $r$ if person $p$ likes restaurant $r$.

Bipartite graphs have many interesting properties. One of the most fundamental is this one:

*An undirected graph is bipartite if and only if it contains no cycles of odd length.*

Intuitively, a bipartite graph contains no odd-length cycles because cycles alternate between the two groups $V_1$ and $V_2$, so any cycle has to have even length. The trickier step is proving that if $G$ is a graph that contains no cycles of odd length, then $G$ has to be bipartite. For now, assume that $G$ has just one connected component; if $G$ has multiple connected components, we can treat each one as a separate graph for the purposes of determining whether $G$ is bipartite. (You don't need to prove this, but I'd recommend taking a minute to check why this is the case.)

Suppose $G$ is a connected, undirected graph with no cycles of odd length. Choose any node $v \in V$. Let
\( V_1 \) be the set of all nodes that are connected to \( v \) by a path of odd length and \( V_2 \) be the set of all nodes connected to \( v \) by a path of even length. (Note that these paths do not have to be simple paths.) Formally:

\[
V_1 = \{ x \in V \mid \text{there is an odd-length path from } v \text{ to } x \} \\
V_2 = \{ x \in V \mid \text{there is an even-length path from } v \text{ to } x \}
\]
i. Given the \( V_1 \) and \( V_2 \) above, prove that \( V_1 \) and \( V_2 \) have no nodes in common.

Remember that there might be multiple different paths of different lengths from \( v \) to some other node \( x \), so be careful not to talk about “the” path between \( v \) and \( x \). Also note that these don’t have to be simple paths.

ii. Using your result from part (i), prove that \( G \) is bipartite.

What do you need to show to prove that every node belongs to one of exactly two sets? Make sure you can point out how you are using the fact that \( G \) is connected and the fact that \( G \) has no cycles of odd length.

Optional Fun Problem 1: Hugs All Around! (1 Point Extra Credit)

There's a party with 137 attendees. Each person is either honest, meaning that they always tell the truth, or mischievous, meaning that they never tell the truth. After everything winds down, everyone is asked how many honest people they hugged at the party. Surprisingly, each of the numbers 0, 1, 2, 3..., and 136 was given as an answer exactly once.

How many honest people were at the party? Prove that your answer is correct and that no other answer could be correct.

Optional Fun Problem 2: How Many Functions Are There? (1 Point Extra Credit)

If \( A \) and \( B \) are sets, we can define the set \( B^A \) to be the set of all functions from \( A \) to \( B \). Formally speaking:

\[
B^A = \{ f \mid f: A \to B \}
\]

Prove that \( |\mathbb{N}| < |\mathbb{N}^\mathbb{N}| \). This shows that \( \aleph_0 < \aleph_0^{\aleph_0} \). Isn’t infinity weird?