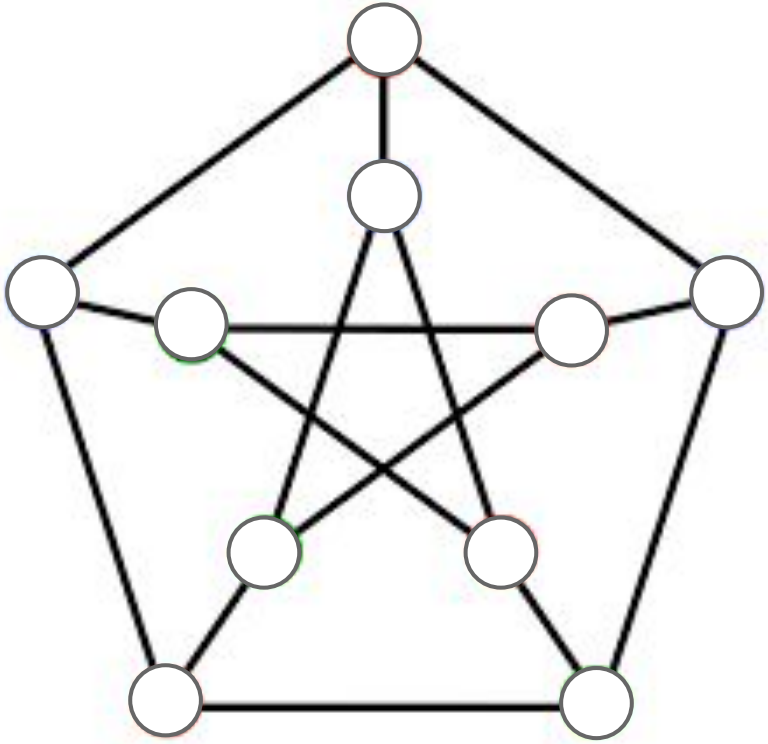
Graph Coloring with Grover's Algorithm: Optimizing Time and Space Efficiencies

Soham Jain[1], Dr. Atul Mantri[2]

Virginia Tech Center for Quantum Information Science and Engineering

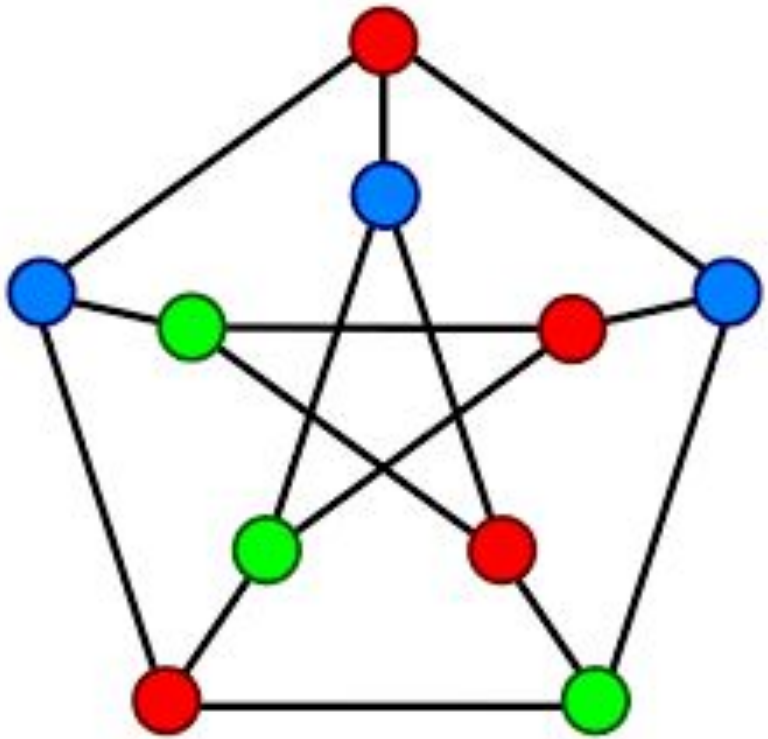
Problem

- Graph Coloring
- Given the following graph and a constraint set:
  - {Red, Green, Blue}
- No two adjacent vertices can be allocated the same color



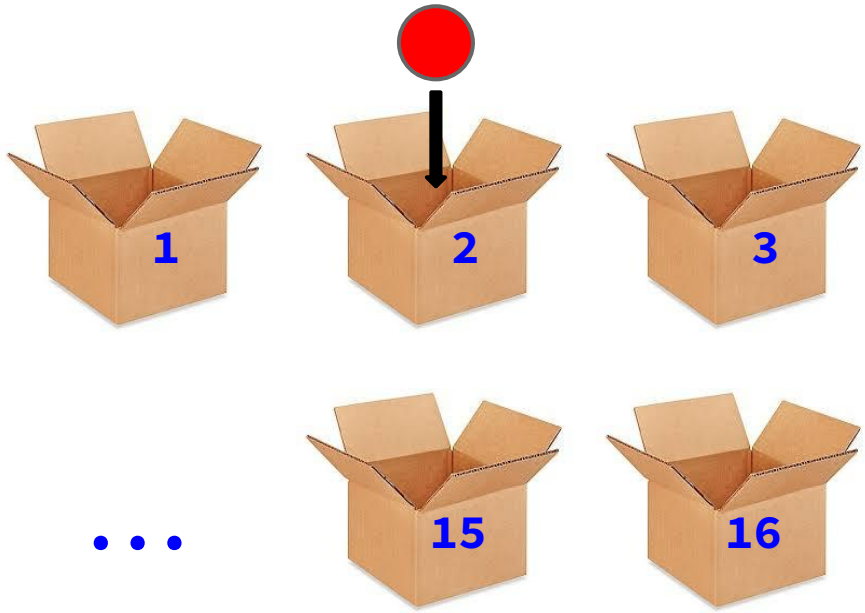
Problem

- Over 59,000 possibilities but only a few solutions
- Computers are often used to solve basic graph coloring problems
    - Significant improvement from trying to do them by hand



Background

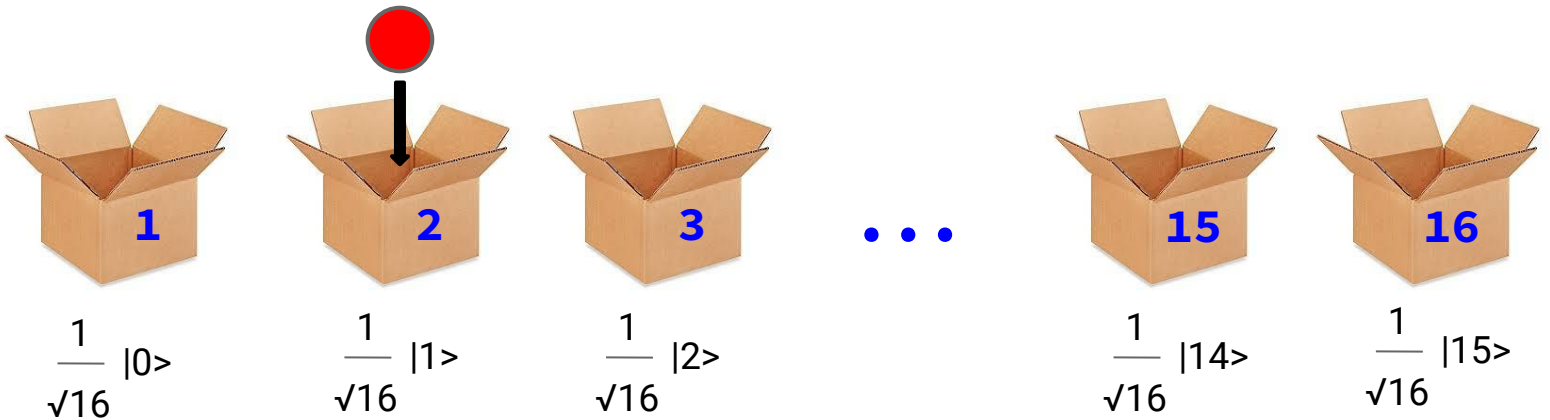
- **Grover's Algorithm**
  - Search algorithm with O(√N) efficiency
  - Requires significantly less memory
- Method 1: Search every box
  - Linear search algorithm
  - O(N) efficiency



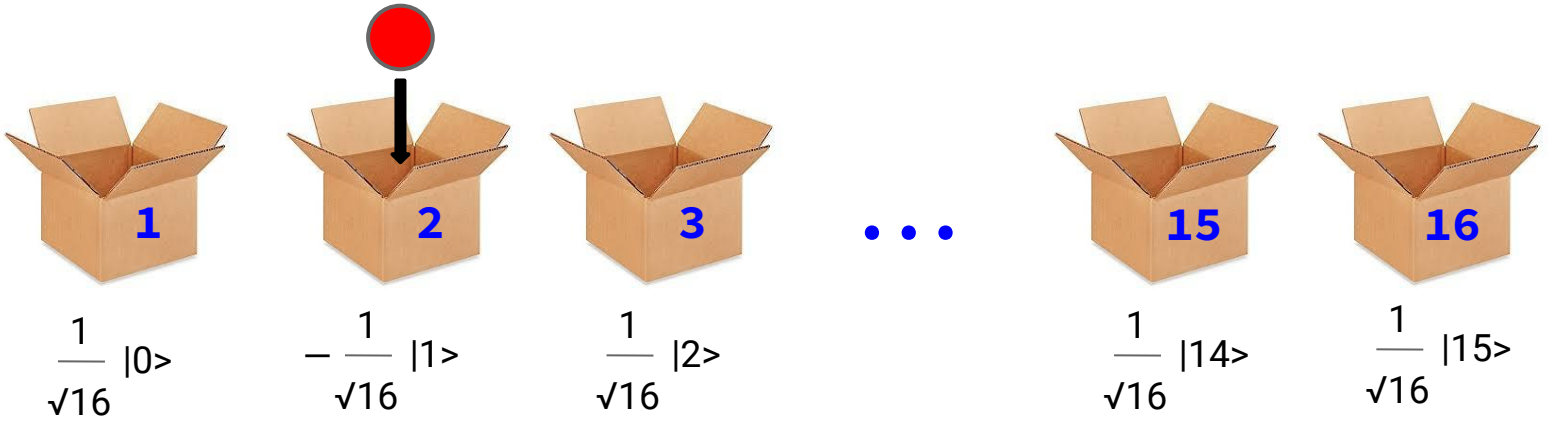
Background

- Method 2: Grover's Algorithm
  - "Super guess" that considers every possibility at once
  - Applies superposition state

$$|\psi\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle$$



Background



- If f(x) = 1 for the correct state |w>, it multiplies the amplitude of that state by −1 (flipping its phase)
- For all other states, where f(x) = 0, the oracle leaves them unchanged

Background

- The oracle knows the correct state by being programmed with a specific function f(x) that encodes the criteria for identifying the target state
- Oracle function is defined as:

x_1	x_2	x_1 XOR x_2
0	0	0
0	1	1
1	0	1
1	1	0

$$\xrightarrow{O_f} |x\rangle \otimes |q \oplus f(x)\rangle$$

$$f(x) = \begin{cases} 0 & \text{if } x \neq u \\ 1 & \text{if } x = u \end{cases}$$

$$|q\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

$$O|\mathbf{x}\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}} \rightarrow |\mathbf{x}\rangle \frac{|f(\mathbf{x}) \oplus 0\rangle - |f(\mathbf{x}) \oplus 1\rangle}{\sqrt{2}}$$

reverse the amplitude if f(x)=1

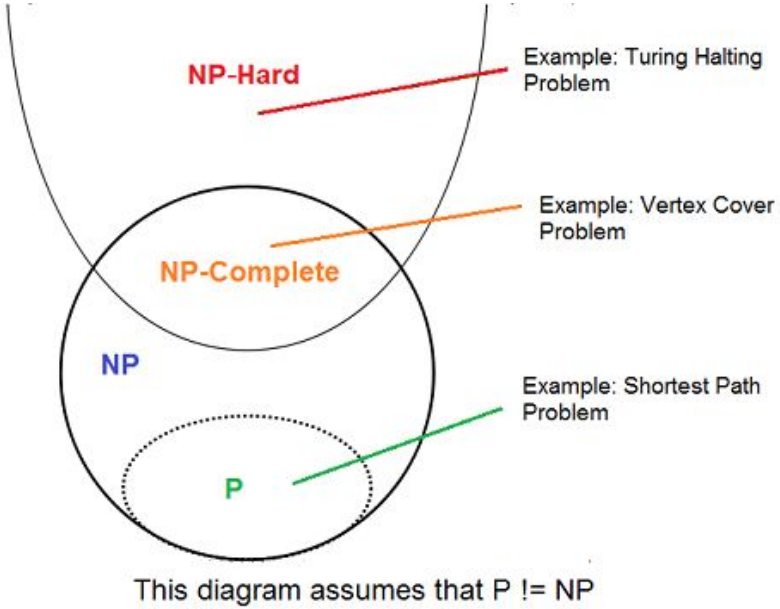
$$\text{if } f(x)=1 \rightarrow |\mathbf{x}\rangle \frac{|1 \oplus 0\rangle - |1 \oplus 1\rangle}{\sqrt{2}} = -|\mathbf{x}\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

$$\text{if } f(x)=0 \rightarrow |\mathbf{x}\rangle \frac{|0 \oplus 0\rangle - |0 \oplus 1\rangle}{\sqrt{2}} = |\mathbf{x}\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

no change

Background: P versus NP Problem

- P is the class of problems that have an algorithm that can be computed in polynomial time
- NP is the class of problems that can be verified in polynomial time, but may take an exponential number of steps to solve

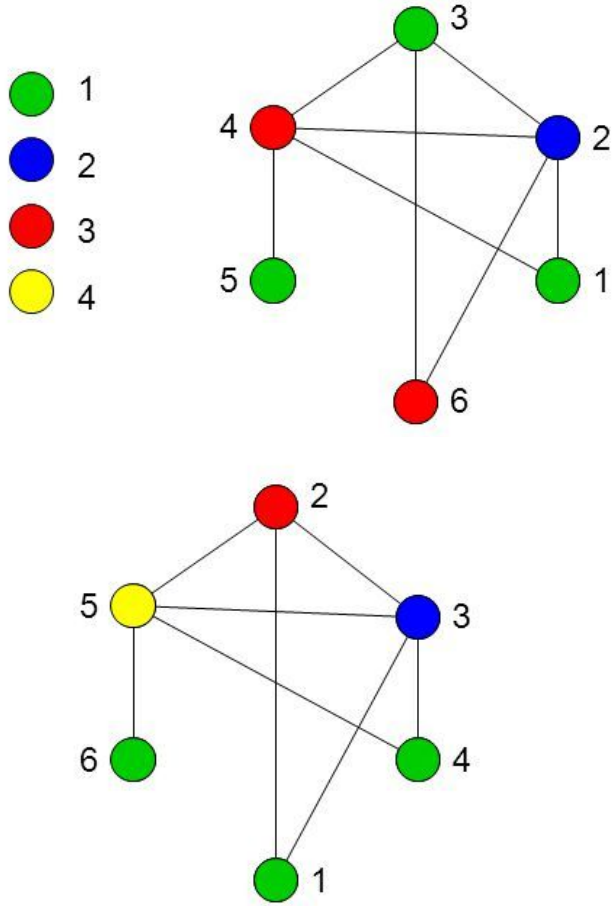


Background: P versus NP Problem

- Grover’s algorithm for graph coloring is an NP-complete problem
- Reduces the search time from O(N) to O(√N)
    - However, even O(√N) is still exponential since N represents an exponentially large number of possible solutions
- The quadratic speedup is not sufficient to solve NP-complete problems in polynomial time

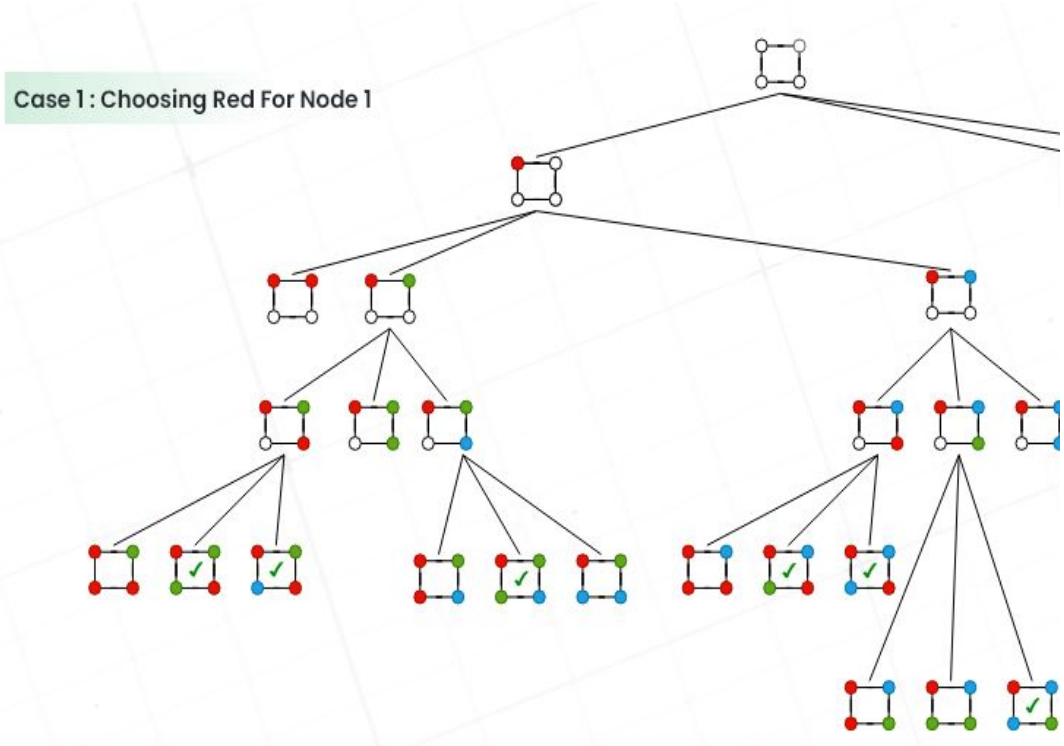
Other Solutions

- **Greedy algorithm**
- Pros:
  - Easy to implement
  - Works well for simple problems
- Cons:
  - Can be quite slow as complexity increases
  - Not always guaranteed



Other Solutions

- **Recursive Algorithms**
- Pros:
  - “Guaranteed” to eventually find a valid configuration
- Cons:
  - High time complexity
  - Memory consumption



Why is Ours Better?

- Existing solutions have a common pattern: time and space complexities
- Grover's algorithm offers a O(√N) time efficiency, compared to O(N) efficiency for linear search
    - In this problem, N is k^50, where k is the number of colors in the constraint set
    - With Grover's algorithm, O(√(k^50)) = O(k^25) efficiency compared to O(k^50) efficiency
- Lower memory requirements compared to classical algorithms

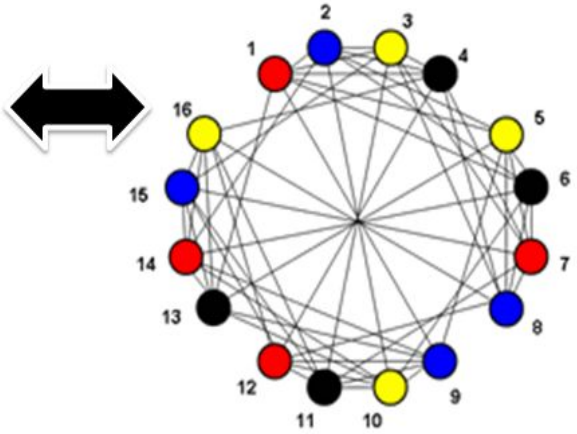
Novelty

- Grover’s algorithm itself is not new and neither is the graph coloring problem
- Grover’s algorithm has not been applied to graph coloring for optimizing efficiency
    - Across multiple programming languages: Qiskit (Python), Q#
- Hybrid integration of quantum algorithm to a classical problem

Impact

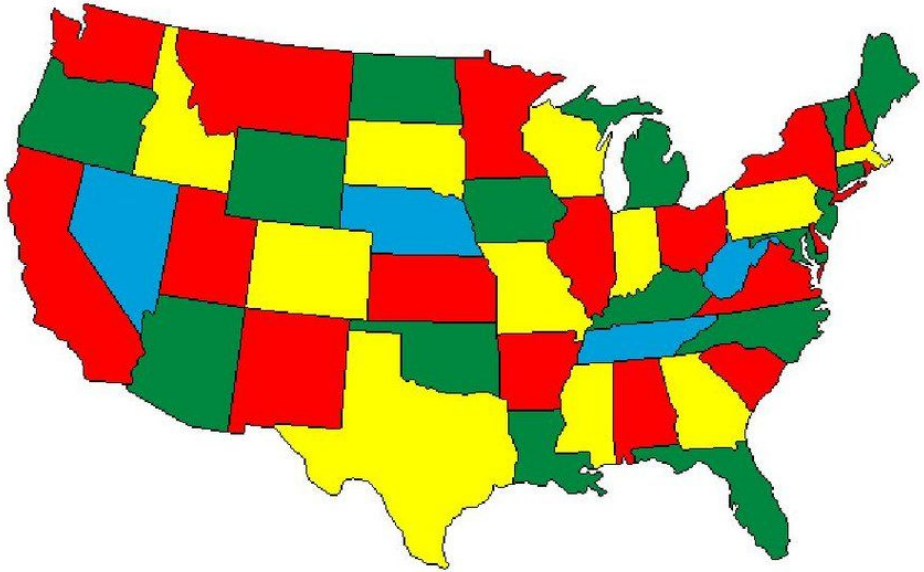
- Most common application is cartography
- Efficient resource allocation via scheduling (e.g. course scheduling, job assignments)
- Solving puzzles and games like Sudoku

1	2	3	4
3	4	1	2
2	3	4	1
4	1	2	3



Method

- Grover’s algorithm across multiple programming languages to solve graph coloring
  - Qiskit (Python), Q#, etc.
- Evaluate efficiencies by running graph coloring on a map of US states



Method: Input

List

```
constraint_set = [“red”, “green”, “blue”, “yellow”]
```

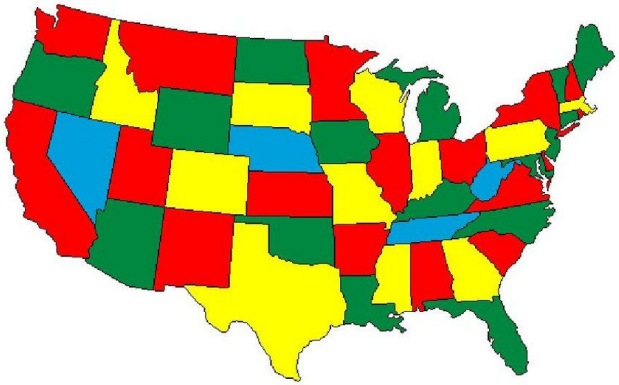
Dictionary

```
input_dict = {
  "Alabama": ["Tennessee", "Georgia", "Florida", "Mississippi"],
  "Alaska": [],
  "Arizona": ["California", "Nevada", "Utah", "Colorado", "New
   Mexico"],
  …
  "Wisconsin": ["Michigan", "Minnesota", "Iowa", "Illinois"],
  "Wyoming": ["Montana", "South Dakota", "Nebraska",
  "Colorado", "Utah", "Idaho"]
}
```

Method: Output

Dictionary

```
output_dict = {
  "Alabama": “red”,
  "Alaska": “green”,
  "Arizona": “blue”,
  …
  "Wisconsin": “green”,
  "Wyoming": “yellow”
}
```

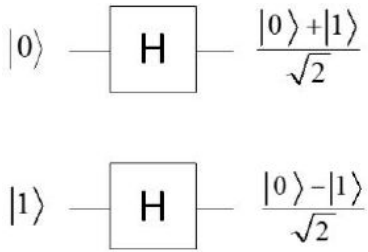


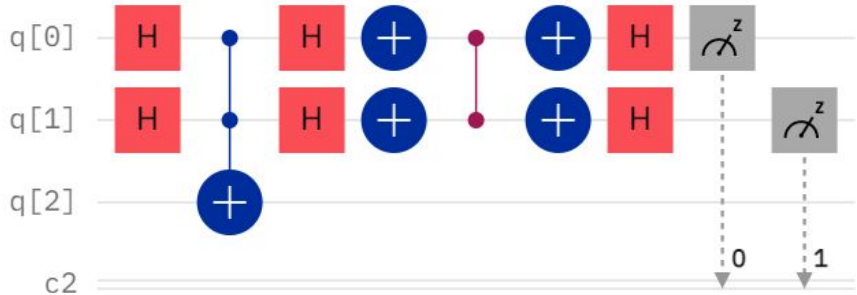
GeoPandas

matplotlib

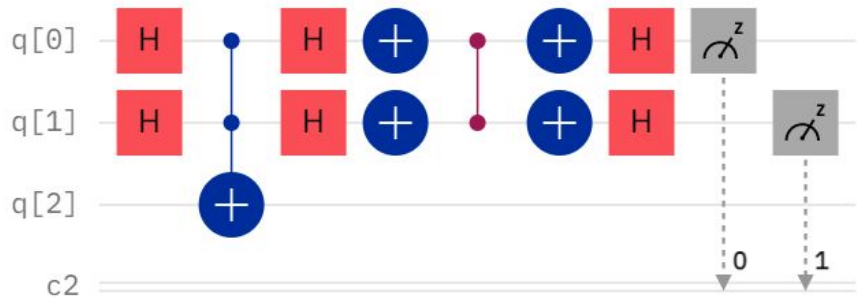
Method: Systems Architecture

- Model will create states for each color in the constraint set: |0>, |1>, |2>, etc.
- Oracle for Grover's algorithm using qiskit libraries
  - These libraries do not create the model themselves, but they have the operations that can be used to create the circuit
- H = Hadamard transform





Method: Pseudocode




```
from qiskit import QuantumCircuit

qc = QuantumCircuit(q, c)

qc.h(q[0])

qc.h(q[1])

qc.cx(q[0], q[2])

qc.cx(q[1], q[2])

qc.h(q[0])

qc.h(q[1])

...

qc.measure(q[0], c2[0])

qc.measure(q[1], c2[1])
```

Results

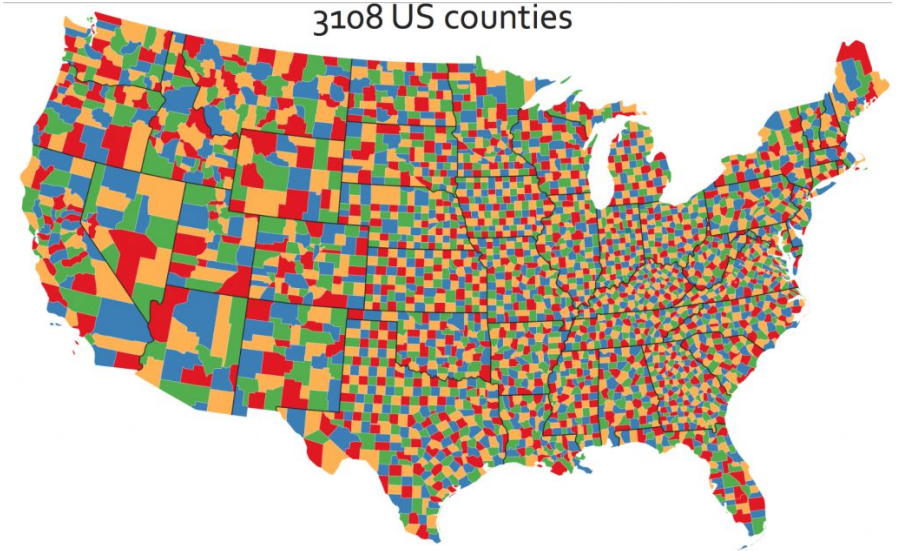
- Will be presented through time and space efficiencies
- Total time taken for the algorithm to execute and return a valid map
    - Will be compared to the time of other algorithms
- Number of bits and qubits used through existing libraries and methods
    - e.g. get_memory_info in Qiskit, PrivateMemorySize64 in Q#, etc.

Limitations

- The time efficiency results are comparable from classical approaches to Grover’s algorithm, but space efficiency results are only comparable between different programming languages
- Due to the limited number of quantum environments right now, it may not be applicable to more complex problems

Conclusion and Future Work

- Will report how results differ from classical approaches (are they an improvement or not?)
- Previous studies suggest that Q# is the most efficient as of now, because of its optimized resource management
    - I want to see if this holds true when applied to the graph coloring problem
- In the future, I graph coloring can be implemented with US counties as well



References

Adams, A. J., Khan, S., Young, J. S., & Conte, T. M. (2024, April 19). *QWERTY: A basis-oriented quantum programming language*. arXiv.org. https://arxiv.org/abs/2404.12603.

Brown, A. R. (2022, December 19). Playing Pool with |ψ>: from Bouncing Billiards to Quantum Search. https://arxiv.org/pdf/1912.02207.

Cornell University. (n.d.). Graph algorithms. https://www.cs.cornell.edu/courses/cs3110/2013sp/supplemental/recitations/rec21-graphs/rec21.html

Graph coloring. Graph Coloring - an overview | ScienceDirect Topics. (n.d.). https://www.sciencedirect.com/topics/computer-science/graph-coloring.

Grover's algorithm. Grover's algorithm | IBM Quantum Learning. (n.d.). https://learning.quantum.ibm.com/tutorial/grovers-algorithm.

Grover's algorithm and amplitude amplification. Grover's Algorithm and Amplitude Amplification - Qiskit Algorithms 0.3.0. (2024, April 10). https://qiskit-community.github.io/qiskit-algorithms/tutorials/06_grover.html

Thanks!

Any Questions?