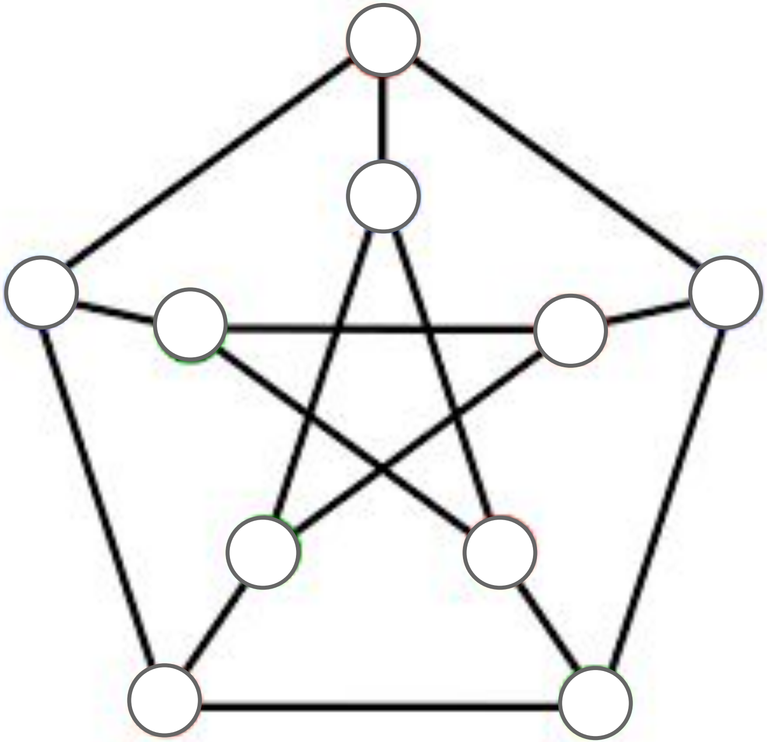
Graph Coloring with Grover's Algorithm: Optimizing Time and Space Efficiencies

Soham Jain
5/21/2025
TJHSST Computer Systems Lab
Yilmaz 1

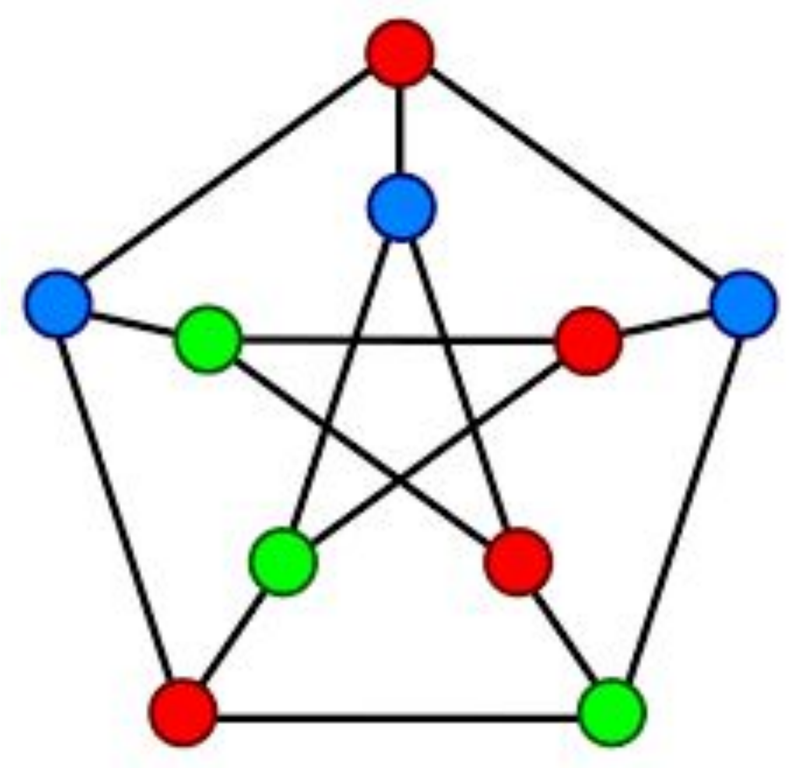
Problem: Graph Coloring

- Given the following graph and a constraint set:
  - {red, green, blue}
- No two adjacent vertices can be allocated the same color



Problem: Graph Coloring

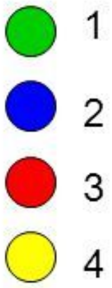
- Over 59,000 possibilities for the graph on the right, but only a few valid solutions
- Exponential time complexity
  - $O(m^V)$, where m is the number of colors and V is the number of vertices
- I use quantum computing to tackle this problem more efficiently



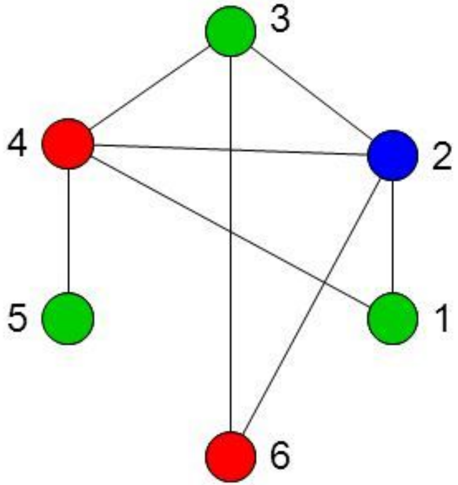
Other Solutions: Greedy Algorithm

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

Color ordering:

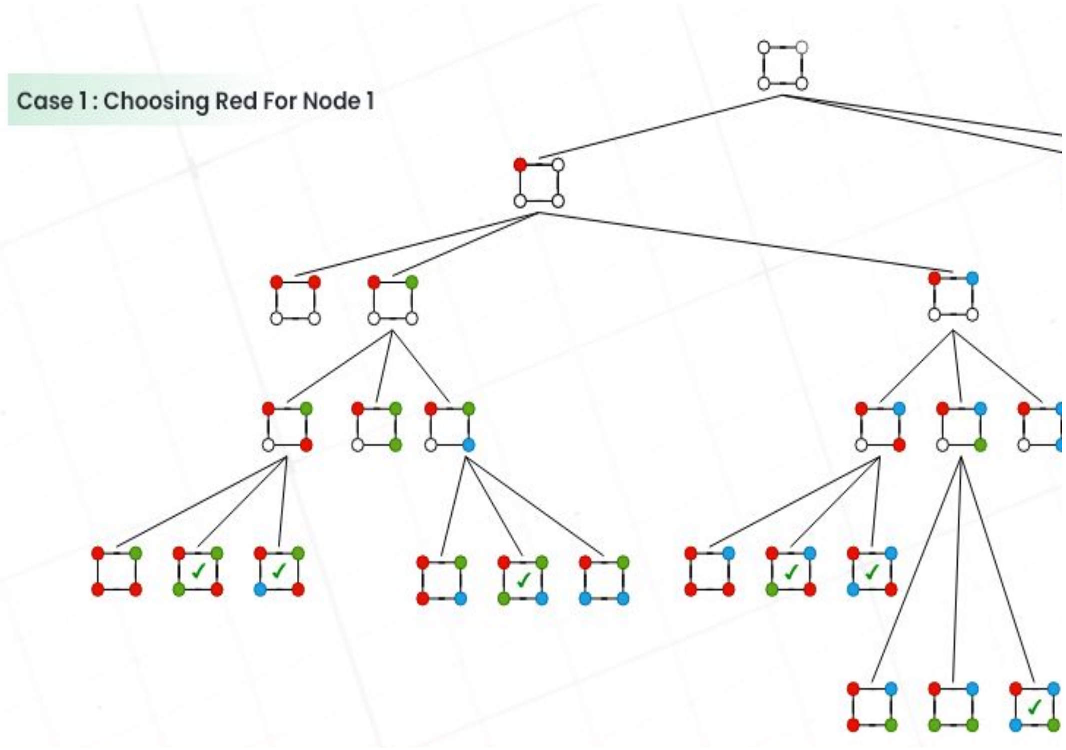


Vertices ordering:



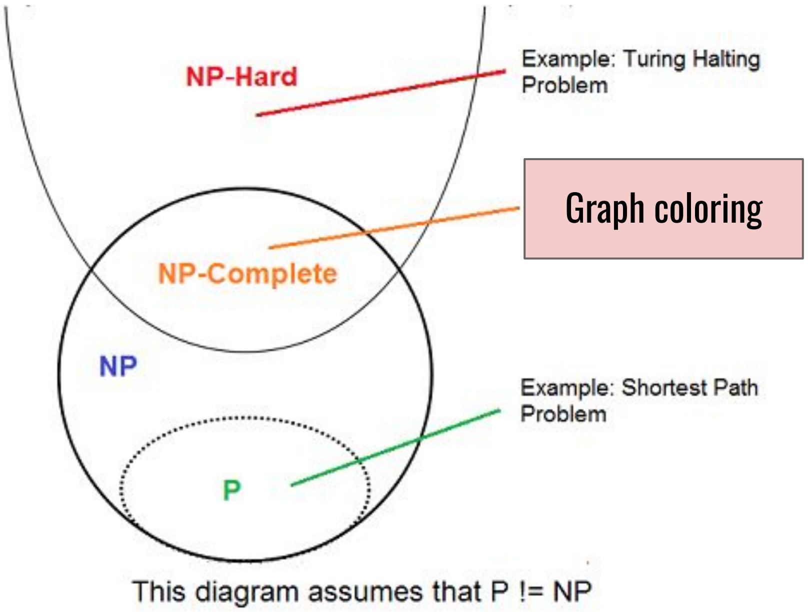
Other Solutions: Recursive Algorithm

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



Background: NP Problem

- NP problems can be verified in polynomial time, e.g. $O(n^2)$, but may take an exponential number of steps, e.g. $O(2^n)$, to solve
- NP-complete represents the hardest problems in NP
- Graph coloring is NP-complete



Background: Grover's Algorithm

- Grover's algorithm is a quantum search algorithm
- It is a heuristic that quadratically speeds up unstructured searches
  - Can be applied to the graph coloring problem
- Uses an oracle, which marks the target state among potential candidates in an unstructured search
  - Diffuser identifies the target by interpreting the oracle's output

Project Goal

- My aim is to implement Grover’s algorithm across multiple programming languages
    - Goal: compare time and space efficiencies across Qiskit and Q#
        - These are common programming languages for quantum computing, but the effect of different implementations on performance is unknown
- Evaluate the results by implementing graph coloring on a map of the 50 US states

Why is Ours Better?

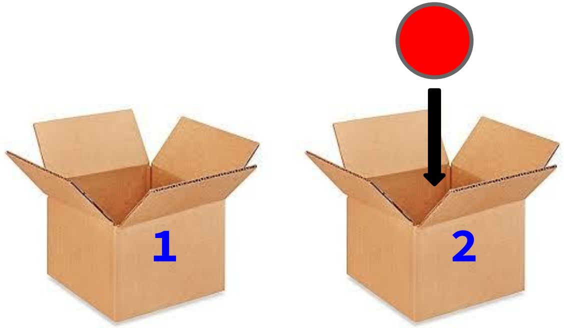
- Existing solutions have a common pattern: time and space complexities
- Grover's algorithm offers an $O(\sqrt{N})$ time efficiency for unstructured search, compared to $O(N)$ efficiency with linear search algorithms
  - In this problem, $N = 4^{50}$, where $m = 4$ and $V = 50$
- Lower memory requirements compared to classical algorithms
- Novelty: Grover's algorithm has not been applied to graph coloring for optimizing efficiency
  - **Across multiple programming languages: Qiskit and Q#**

Why is Quantum More Efficient?

- Both classical and quantum approaches consider all possible colorings, but the key difference lies in how they do this and how quickly they zero in on the right answer
  - Classical recursion explores each possible coloring step by step, backtracking when constraints are violated
  - Grover's algorithm, on the other hand, uses quantum parallelism to explore all colorings simultaneously in a superposition of states.
    - Instead of checking one by one, the valid solution "pops out"

Background: Linear Search Algorithm

- Method 1: Search every box
  - O(N) efficiency
  - In the worst case, you would have to search N = 2 boxes to find the red ball



Background: Grover’s Algorithm

- Method 2: Grover’s Algorithm
    - “Super guess” that considers every possibility at once
    - Applies superposition state

state

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

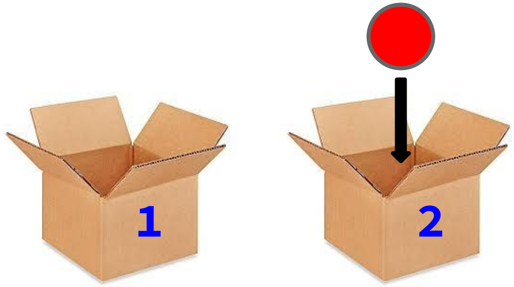
N = # of boxes

index

$\frac{1}{\sqrt{2}}|0\rangle$ $\frac{1}{\sqrt{2}}|1\rangle$

Background: Grover's Algorithm

- If it is the correct state, the oracle multiplies the amplitude by −1
- For all other states, the oracle leaves them unchanged
- Diffuser identifies the correct state by measuring amplitudes



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

Background: Grover's Algorithm

- The oracle is programmed with a specific function, f(x), which 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

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

XOR

Current state

Tensor product (quantum multiplication)

Current state

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

Correct state

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

Constant

Background: Grover's Algorithm

- The oracle is programmed with a specific function, f(x), which 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

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

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

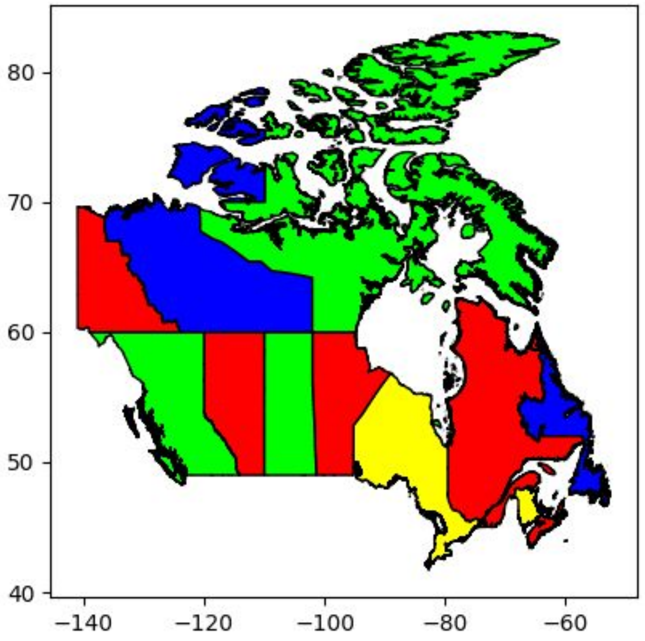
no change

Grover’s Algorithm Time Complexity

- Reduces unstructured search time (box problem) from $O(N)$ to $O(\sqrt{N})$
    - However, even $O(\sqrt{N})$ is still exponential since N represents an exponentially large number of possible solutions
- Grover’s algorithm offers a quadratic speedup
    - But this speedup is not sufficient to solve NP-complete problems in polynomial time

Related Work

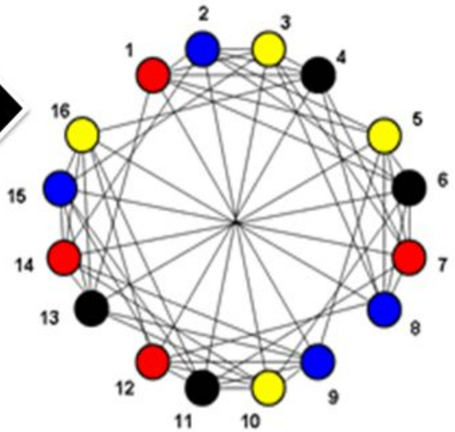
- Kjer (2023) used Grover's algorithm for map coloring 13 provinces in Canada
  - My project involves map coloring with significantly more vertices (V = 50)
  - I also evaluated this solution across multiple languages, instead of just Qiskit
- Clerc (2023) found that graph coloring is polynomial for K < 3 or K = 3 in some cases, where K is the length of the constraint set
  - However, K = 4 in this problem, so it is still NP-complete



Impact

- Graph coloring is a constraint satisfaction problem (CSP)
- Our findings can be generalized for other CSPs like course scheduling, Sudoku, etc.
- Most common application is cartography

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



Method: Input

List

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

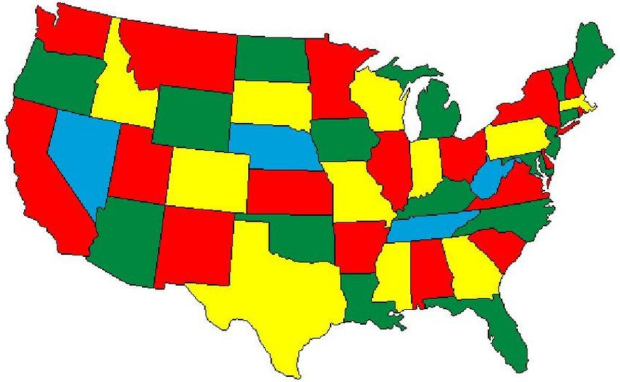
Dictionary

```
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”
}
```

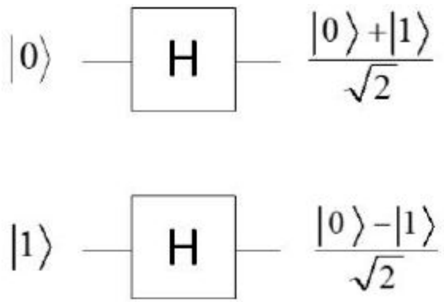


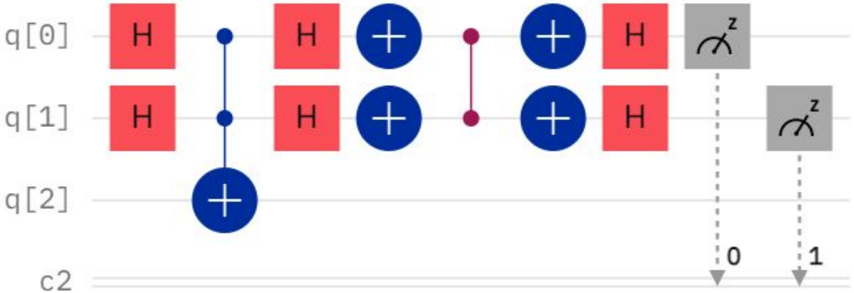
GeoPandas

matplotlib

Method: Grover’s Algorithm

- Oracle and diffuser for Grover’s algorithm using Qiskit and Q# 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: Diffuser and Oracle

```python
def diffuser(n_qubits):
    circuit = QuantumCircuit(n_qubits)
    circuit.h(range(n_qubits))
    circuit.x(range(n_qubits))
    circuit.h(n_qubits - 1)
    circuit.mcx(list(range(n_qubits - 1)), n_qubits - 1)
    circuit.h(n_qubits - 1)
    circuit.x(range(n_qubits))
    circuit.h(range(n_qubits))
    return circuit

def oracle(vertices, edges, num_colors):
    color_bits = int(np.log2(num_colors))
    n_qubits = len(vertices) * color_bits
    grover_circuit = QuantumCircuit(n_qubits)
    grover_circuit.h(range(n_qubits))
    createCircuitConnection(vertices, edges, num_colors, grover_circuit)
    grover_circuit.h(range(n_qubits))
    return grover_circuit
```

Qiskit

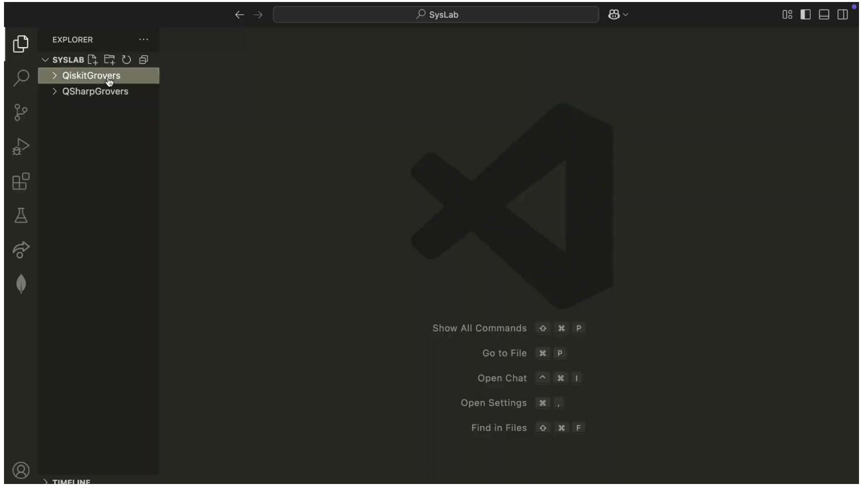
```
operation Oracle(qs : Qubit[]) : Unit is Adj {
    within {
        for qubit in qs {
            X(qubit);
        }
    } apply {
        Z(qs[0]);
        for qubit in qs {
            X(qubit);
        }
    }
}
```

Circuit

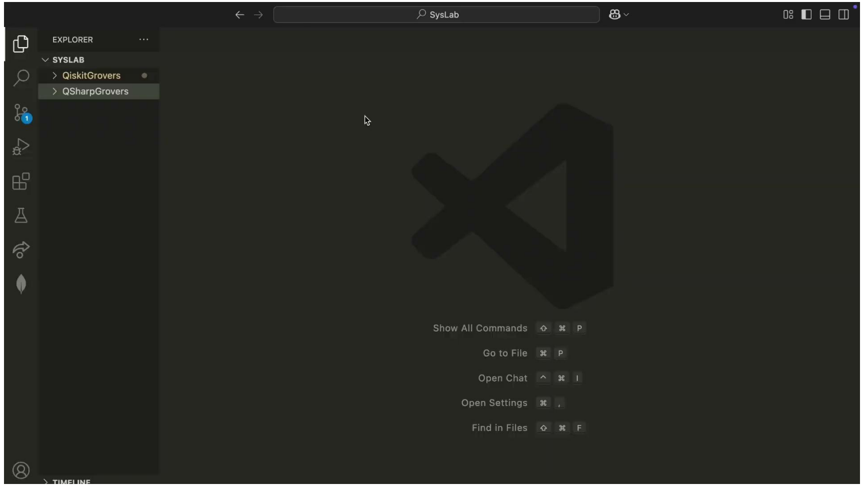
```
operation Diffuser(qs : Qubit[]) : Unit is Adj {
    within {
        for qubit in qs {
            H(qubit);
        }
        for qubit in qs {
            X(qubit);
        }
    } apply {
        Controlled Z(Most(qs), Tail(qs));
        for qubit in qs {
            X(qubit);
        }
        for qubit in qs {
            H(qubit);
        }
    }
}
```

Q#

Demo



Qiskit



Q#

Results

- Presented through time and space efficiencies
- Total time taken for the algorithm to execute and return a valid map
  - Compared between programming languages
- Space efficiency was calculated through quantifying the circuit depth

Results

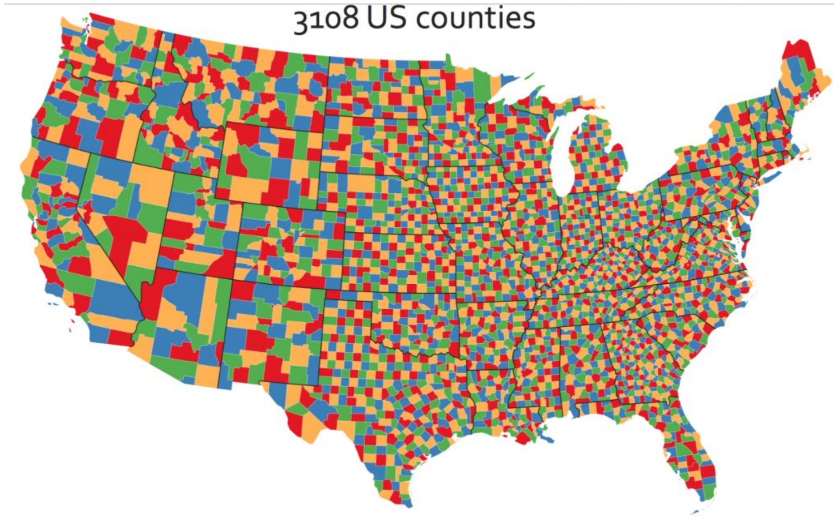
	Average Time (20 trials)	Circuit depth
Qiskit	14.94 seconds	34-42 gates
Q#	34.78 seconds	26-38 gates
Recursive (Python)	43.75 seconds	N/A

Conclusion and Future Work

- Qiskit was the most efficient programming language in terms of time, but Q# had better space efficiency
    - This is likely because Qiskit’s simulator (AerSimulator) is highly optimized, while Q# prioritizes quantum memory management
- In the future, graph coloring can be implemented with US counties
    - Furthermore, this problem can be tested on more programming languages (Cirq, Ocean, etc.) and other constraint satisfaction problems (Sudoku, course scheduling, etc.)

Limitations

- Time efficiency results are comparable with classical approaches
  - However, space efficiency is only comparable between different programming languages
- Due to the limited number of quantum environments right now, problems with more vertices and a larger constraint set cannot be solved with Grover's algorithm



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

Maurice Clerc. A general quantum method to solve the graph K-colouring problem. 2023. https://hal.science/hal-02891847/document

Nathan Kjer. (2023, January 19). Quantum Computing: Map coloring via grover's algorithm. https://nathankjer.com/grovers-algorithm/

https://davidbkemp.github.io/animated-qubits/grover.html

Thanks!

Any Questions?