**Introduction:** In this report, we present a comparative analysis between D-Wave's Ising algorithm and Qiskit's gate-based solution for solving the Ising model problem. The aim is to provide insights into the key elements of each approach, facilitating a better understanding of quantum computing platforms for those unfamiliar with them.

**Methodology:** We conducted a comprehensive review of the literature on D-Wave's Ising algorithm and Qiskit's gate-based solution. The analysis focused on identifying and comparing the main elements of each approach, including algorithm structure, hardware/software requirements, optimization approaches, and application flexibility.

| **Element** | **D-Wave's Ising Algorithm** | **Qiskit's Gate-Based Solution** |
| --- | --- | --- |
| Algorithm Structure | Utilizes quantum annealing | Utilizes gate-model quantum computation |
| Hardware Requirements | Requires access to D-Wave quantum annealers | Compatible with various quantum devices |
| Software Requirements | Access to D-Wave's quantum computing platform | Qiskit library and compatible backend |
| Ising Model Representation | Encoded directly into hardware as Ising spins | Encoded as qubits with appropriate gates |
| Quantum Gates Used | N/A | Utilizes gates such as Hadamard, CNOT, etc. |
| Optimization Approach | Energy minimization based on Ising model | Quantum circuit manipulation for optimization |
| Supported Problem Size | Limited by the number of qubits in the hardware | Dependent on available qubits and circuit complexity |
| Noise and Error Handling | D-Wave's annealing calibration and error correction | Qiskit's error mitigation techniques like error correction codes, noise models, etc. |
| Application Flexibility | Primarily suited for optimization problems | Versatile, suitable for various quantum algorithms |
| Programming Interface | D-Wave's proprietary tools and SDK | Qiskit's Python-based programming interface |

1. **PENQASM 2.0**: This line specifies the version of the OpenQASM language being used.
2. **include "qelib1.inc"**: This line includes the standard quantum gate definitions from the qelib1 library. This library provides common quantum gates such as the Hadamard gate (**h**), controlled-Z gate (**cz**), and single-qubit rotation gates.
3. **qreg q[3];**: This line declares a quantum register named **q** containing 3 qubits.
4. **creg c[3];**: This line declares a classical register named **c** containing 3 classical bits. Classical registers are used to store measurement outcomes.
5. **h q[0]; h q[1]; h q[2];**: These three lines apply the Hadamard gate (**h**) to each qubit in the quantum register **q**. The Hadamard gate creates superposition by putting the qubits into an equal superposition of the |0⟩ and |1⟩ states.
6. **rz(-2.0) q[1]; cz q[0], q[1];**: These two lines apply a rotation around the Z-axis (controlled by the constant **-2.0**) to the second qubit (**q[1]**) and then apply a controlled-Z (CZ) gate between the first and second qubits (**q[0]** and **q[1]**). The CZ gate performs a conditional phase shift on the target qubit (**q[1]**) depending on the state of the control qubit (**q[0]**).
7. **rz(-2.0) q[2]; cz q[1], q[2];**: Similar to the previous lines, these two lines apply a rotation around the Z-axis to the third qubit (**q[2]**) and then apply a CZ gate between the second and third qubits (**q[1]** and **q[2]**).
8. **rz(1.0) q[0]; rz(1.0) q[1]; rz(1.0) q[2];**: These three lines apply rotations around the Z-axis to each qubit in the quantum register **q** with the constant value **1.0**. These rotations represent the application of a local magnetic field to each qubit.
9. **measure q[0] -> c[0]; measure q[1] -> c[1]; measure q[2] -> c[2];**: These three lines perform measurements on each qubit in the quantum register **q** and store the measurement outcomes in the corresponding classical bits in the classical register **c**.

In summary, this OpenQASM code prepares the qubits in superposition states using Hadamard gates, applies interactions between neighboring qubits, and applies local magnetic fields. Finally, it measures the qubits to obtain measurement outcomes.

1. **pplying Hadamard gates**: The Hadamard gates (**h**) are applied to put the qubits into superposition states. In the Ising model context, this step helps in exploring different possible spin configurations simultaneously, which is crucial for optimization problems.
2. **Applying ZZ interactions**: The **rz** gates followed by **cz** gates represent ZZ interactions between neighboring qubits in the Ising model. In the Ising model, these interactions correspond to the coupling strength (**J**) between spins in the lattice. By applying these gates, we simulate the interaction between spins, which is essential for understanding the system's energy landscape.
3. **Applying local magnetic fields**: The **rz** gates with constant values represent the application of local magnetic fields to each qubit. In the Ising model, these local fields correspond to an external magnetic field (**h**) applied to each spin. These fields can influence the orientation of individual spins and affect the overall energy of the system.
4. **Performing measurements**: Finally, measurements are performed on each qubit to obtain measurement outcomes, which are then stored in classical bits. These measurement outcomes provide information about the spin configurations of the system, allowing us to analyze its properties and potentially find optimal solutions for optimization problems modeled using the Ising model.