1. Постановка задачи

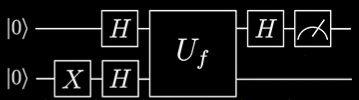
Цель алгоритма Дойча: Определить, является ли функция f: {0,1} -> {0,1} постоянной

(f(0) = f(1)) или сбалансированной (f(0) ≠ f(1)).

Преимущество квантового алгоритма:

Классический подход требует 2 вычисления функции f, чтобы определить её тип, в то время как квантовый алгоритм Дойча решает задачу за 1 вызов функции, используя суперпозицию и интерференцию кубитов.

1. Схема алгоритма



1. Инициализируем два кубита
2. К первому применяем гейт Адамара, ко второму инверсию, а затем гейт Адамара.
3. Применяем оракул U к обоим кубитам
4. Применяем гейт Адамара к первому кубиту
5. Измеряем первый кубит
6. Код на OpenQASM

OPENQASM 2.0;

include "qelib1.inc";

// Объявление квантового регистра (2 кубита) и классического регистра (1 бит)

qreg q[2];

creg c[1];

// Инициализация второго кубита в |1⟩

x q[1];

// Применение гейтов Адамара к обоим кубитам

h q[0];

h q[1];

// Оракул U\_f (f(x) = x) — сбалансированная функция

cx q[0], q[1]; // CNOT: q[1] = q[1] XOR f(q[0])

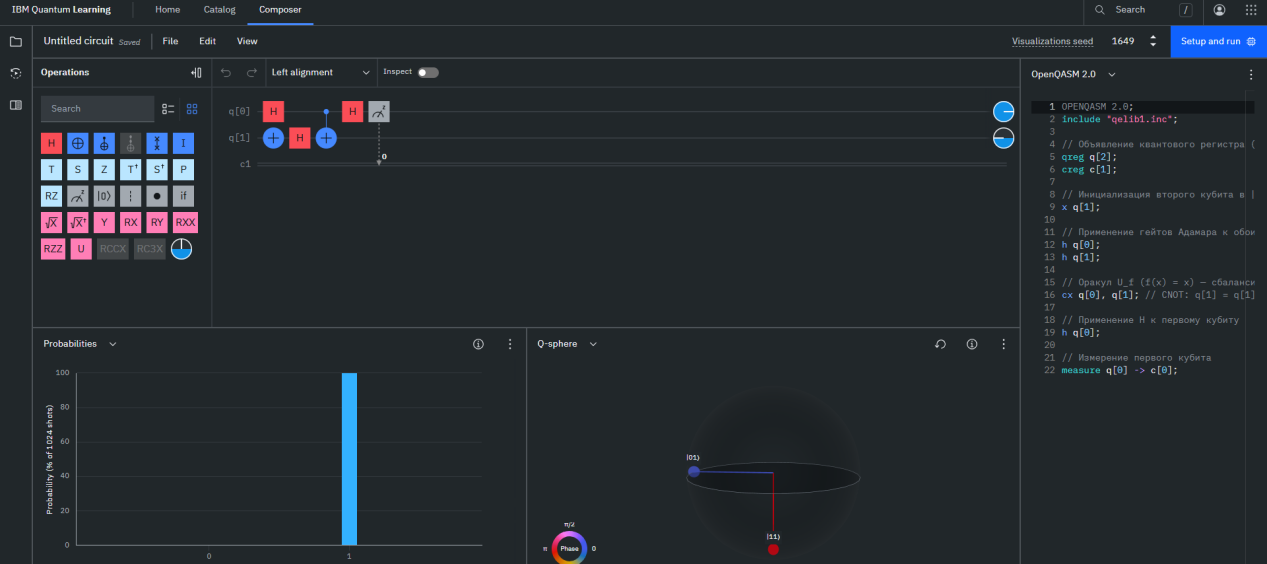
// Применение H к первому кубиту

h q[0];

// Измерение первого кубита

measure q[0] -> c[0];

https://quantum-circuit.com/qconvert



Код на qiskit с учётом шумов:

# === Импорты ===

from qiskit import QuantumCircuit, transpile

from qiskit\_aer import AerSimulator

from qiskit\_ibm\_runtime.fake\_provider import FakeJakartaV2 as FakeJakarta

# === Построение схемы ===

qc = QuantumCircuit(2, 1)

qc.x(1) # подготовка |1⟩ на втором кубите

qc.h([0, 1]) # суперпозиция обоих кубитов

qc.cx(0, 1) # оракул f(x)=x

qc.h(0) # интерференция перед измерением

qc.measure(0, 0)

# === Симуляция с шумом устройства ===

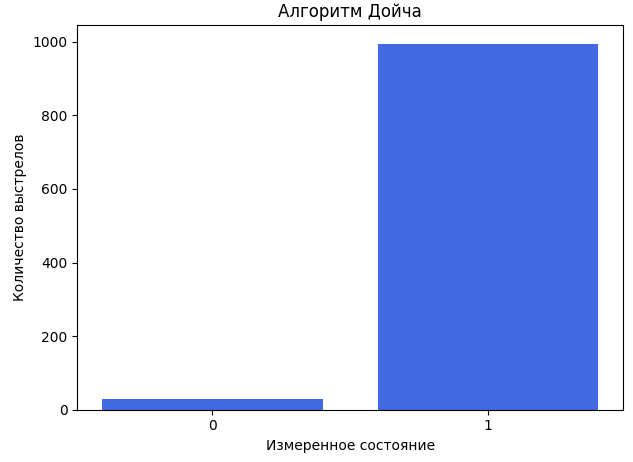
backend = FakeJakarta()

sim = AerSimulator.from\_backend(backend)

t\_qc = transpile(qc, backend=backend, optimization\_level=3)

counts = sim.run(t\_qc, shots=1024).result().get\_counts()

print(counts)



Алгоритм Дойча демонстрирует фундаментальное преимущество квантовых вычислений над классическими, решая задачу определения типа функции

f:{0,1}→{0,1} за один вызов функции, в отличие от классического подхода, требующего двух вычислений. Это достигается за счёт свойств квантовой механики, таких как суперпозиция, интерференция и квантовый параллелизм.

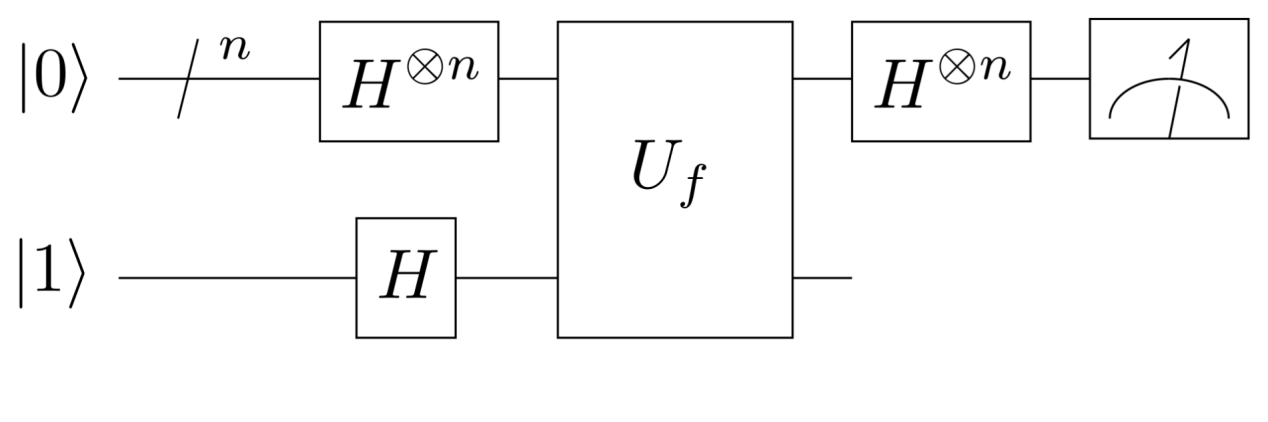
2.Постановка задачи

Цель алгоритма Дойча-Йожи: является ли функция f: {0,1}^n -> {0,1} постоянной (значение f(x) одинаково для всех x) или сбалансированной (значение f(x) равно 0 ровно для половины входов и 1 для другой половины).

Преимущество квантвого алгоритма:

В то время как классический подход требует O(2^n) вычислений функии f, квантовый алгоритм решает задачу за 1 вызов функции, используя квантовый параллелизм и интерференцию.

1. Квантовая схема алгоритма



3)OPENQASM 2.0;

include "qelib1.inc";

qreg q[3]; // 2 входных кубита + 1 вспомогательный

creg c[2]; // Результаты измерения входных кубитов

// Инициализация вспомогательного кубита в |1⟩

x q[2];

h q[2];

// Суперпозиция входных кубитов

h q[0];

h q[1];

// Оракул U\_f (f(x) = x0 ⊕ x1)

cx q[0], q[2];

cx q[1], q[2];

z q[2]; // Фазовый сдвиг для сбалансированной функции

// Интерференция

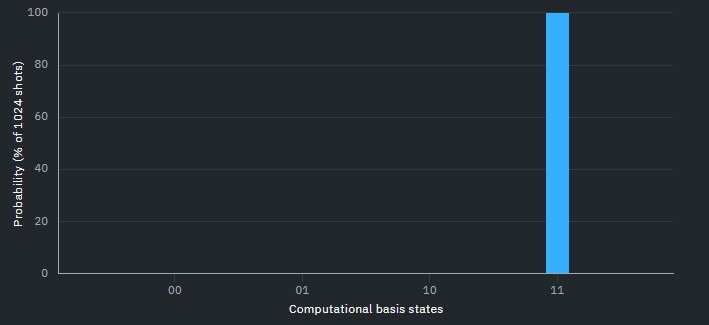
h q[0];

h q[1];

// Измерение входных кубитов

measure q[0] -> c[0];

measure q[1] -> c[1];



Код на qiskit с учётом шумов:

# === Импорты ===

from qiskit import QuantumCircuit, transpile

from qiskit\_aer import AerSimulator

from qiskit\_ibm\_runtime.fake\_provider import FakeJakartaV2 as FakeJakarta

from qiskit.circuit.library import ZGate

# === Построение схемы ===

qc = QuantumCircuit(3, 2)

qc.x(2); qc.h(2) # подготовка |−⟩ на вспомогательном кубите

qc.h([0, 1]) # суперпозиция входных кубитов

qc.cx(0, 2); qc.cx(1, 2) # оракул f(x0,x1)=x0⊕x1

qc.append(ZGate(), [2]) # фазовый сдвиг

qc.h([0, 1]) # интерференция перед измерением

qc.measure([0, 1], [0, 1])

# === Симуляция с шумом устройства ===

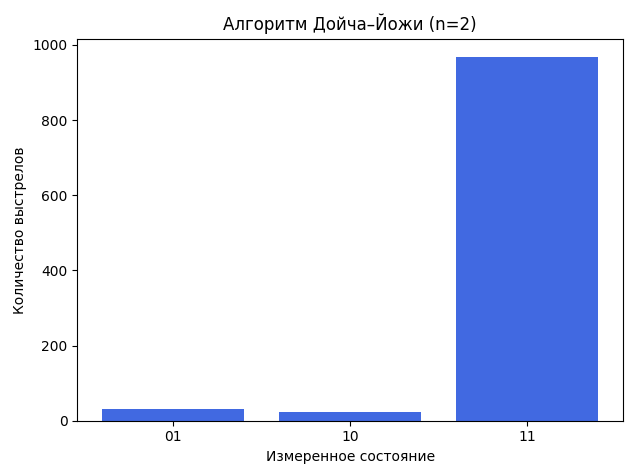
backend = FakeJakarta()

sim = AerSimulator.from\_backend(backend)

t\_qc = transpile(qc, backend=backend, optimization\_level=3)

counts = sim.run(t\_qc, shots=1024).result().get\_counts()

print(counts)



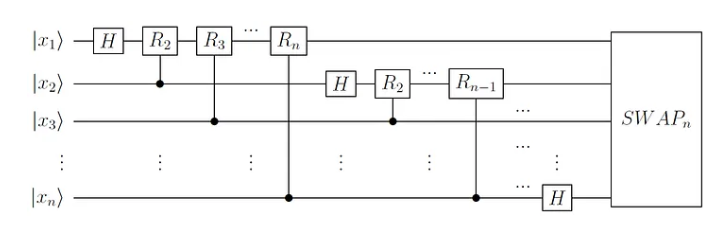
Алгоритм Дойча-Йожи демонстрирует экспоненциальное ускорение, решая задачу определения типа функции для n входов за 1 квантовый запрос (вместо O(2^n) классических вычислений), используя суперпозицию и интерференцию для одновременного анализа всех входов, и служит основой для ключевых квантовых алгоритмов, таких как Шора и Гровера.

1. Постановка задачи:

Цель алгоритма: Выполнить дискретное преобразование Фурье над квантовым состоянием.

Применяется алгоритм в качестве ключевого компонента для факторизации чисел в алгоритме Шора, а также используется в квантовой симуляции и обработке сигналов.

2)Квантовая схема



1. Код

OPENQASM 2.0;

include "qelib1.inc"; // Подключение стандартной библиотеки гейтов

qreg q[3]; // 3 кубита

creg c[3]; // 3 классических бита для измерения

// Применение QFT

// Шаг 1: Гейты Адамара и управляемые фазовые вращения

h q[0];

cu1(pi/2) q[1], q[0]; // Управляемое вращение на π/2 (CPHASE)

cu1(pi/4) q[2], q[0]; // Управляемое вращение на π/4

h q[1];

cu1(pi/2) q[2], q[1]; // Управляемое вращение на π/2

h q[2];

// Шаг 2: SWAP-гейты для коррекции порядка кубитов

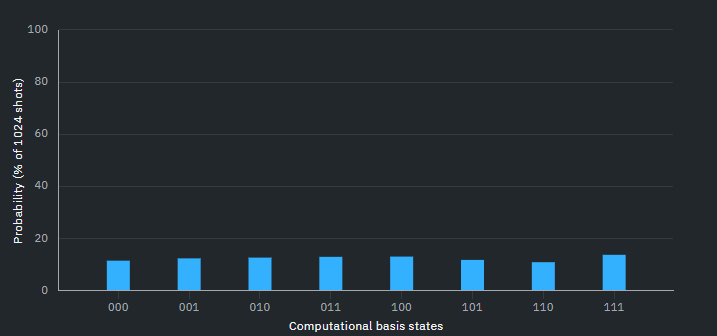
swap q[0], q[2]; // Меняем местами первый и последний кубиты

// Измерение

measure q[0] -> c[0];

measure q[1] -> c[1];

measure q[2] -> c[2];



Код на qiskit с учётом шумов:

# === Импорты ===

from qiskit import QuantumCircuit, transpile

from qiskit\_aer import AerSimulator

from qiskit\_ibm\_runtime.fake\_provider import FakeJakartaV2 as FakeJakarta

from qiskit.circuit.library import QFT

# === Построение схемы ===

qc = QuantumCircuit(3, 3)

qc.h(0) # подготовка |+00⟩

qc.append(QFT(3, do\_swaps=True), range(3))

qc.measure(range(3), range(3))

# === Симуляция с шумом устройства ===

backend = FakeJakarta()

sim = AerSimulator.from\_backend(backend)

t\_qc = transpile(qc, backend=backend, optimization\_level=3)

counts = sim.run(t\_qc, shots=1024).result().get\_counts()

print(counts)



Алгоритм квантового преобразования Фурье (QFT) обеспечивает экспоненциальное ускорение по сравнению с классическим аналогом, являясь ключевым компонентом алгоритмов Шора и квантовой симуляции, но требует устойчивости к шумам и коррекции ошибок для реализации на современных квантовых устройствах.