Der Bernstein-Vazirani-Algorithmus – aus der Informatikperspektive

Einleitung

Stell dir vor, du bist in einer Spielhalle und entdeckst einen geheimnisvollen Spielautomaten. So funktioniert er: Du hast drei Münzen – einige 10-Cent-Münzen und einige 20-Cent-Münzen. Der Automat erlaubt dir, diese Münzen in verschiedenen Kombinationen einzuwerfen. Als Ergebnis bekommst du entweder nichts oder genau 1 Euro. Dein Ziel ist es, die exakte versteckte Kombination aus 10-Cent- und 20-Cent-Münzen herauszufinden, die den Dollar-Jackpot auslöst.

Dieses Spiel ist eine anschauliche Analogie für den **Bernstein-Vazirani-Algorithmus**, der ein ähnliches Problem löst – allerdings mit einem Quanten-Algorithmus! So wie der Spielautomat eine versteckte Regel besitzt, die von deiner Münzkombination abhängt, geht es bei diesem Algorithmus darum, ein geheimes Muster (eine versteckte Folge aus 0 und 1) zu entdecken, das das Verhalten einer Funktion bestimmt. Die Herausforderung besteht darin, dies möglichst effizient zu tun.

Spielregeln:

1. Die Schülerinnen und Schüler erhalten drei Münzen, dargestellt als Bits (0 für die 10-Cent-Münze und 1 für die 25-Cent-Münze)

2. Der Spielautomat (Lehrkraft) kennt die geheime Kombination (versteckte Bitfolge) und überprüft die eingegebene Kombination. Er gibt zurück: 0 → kein Gewinn oder 1 → Gewinn (1 Euro).

3. Ziel: Die Schülerinnen und Schüler sollen die versteckte Regel (Bitfolge) herausfinden.

Mathematische Beschreibung des Problems

Stell dir vor, du hast eine Blackbox, die durch eine Funktion f_a definiert wird. Die Eingabe in die Box (bzw. in die Funktion) ist eine Binärzahl mit n Bits. Die Ausgabe ist entweder 0 oder 1. Die Funktion f_a wird über eine festgelegte geheime Binärzahl $a = (a_1, a_2, \ldots, a_n)$ definiert, wobei jedes a_i 0 oder 1 ist.

Um die Ausgabe $f_a(x)$ zu berechnen, machen wir die folgenden Schritte[1]:

- Die Eingabe besteht aus n Bits $x = (x_1, x_2, \ldots, x_n)$, wobei x_i jeweils 0 oder 1 ist.
- Multipliziere jedes Eingabebit mit dem entsprechenden Bit aus a: $a_1 \cdot x_1 + a_2 \cdot x_2 + \ldots + a_n \cdot x_n$
- Überprüfe das Ergebnis der Summe:
  - Falls die Summe gerade ist, gibt die Funktion 0 zurück.
  - Falls die Summe ungerade ist, gibt die Funktion 1 zurück.

ENTNOMMEN AUS QUANTEN-COMPUTING IM MINT-UNTERRICHT
Ein Projekt von Science on Stage Deutschland e.V.

Aufgabe 1

Berechne die Ausgaben $\boldsymbol{f_a(x)}$ für $\boldsymbol{a} = (\mathbf{1},\mathbf{0},\mathbf{1})$ (also $\boldsymbol{n} = \mathbf{3}$) anhand der Tabelle:

Input x	$a \cdot x = a_1 \cdot x_1 + a_2 \cdot x_2 + a_3 \cdot x_3$	ungerade	gerade	Output $f_a(x)$
$(0,0,0)$	$1 \cdot 0 + 0 \cdot 0 + 1 \cdot 0 = 0$		x	0
$(0,0,1)$	$1 \cdot 0 + 0 \cdot 0 + 1 \cdot 1 = 1$	x		1
$(0,1,0)$				
$(1,0,0)$				
$(0,1,1)$				
$(1,0,1)$				
$(1,1,0)$				
$(1,1,1)$				

Analysiere die Ergebnisse: Wie viele und welche Kombinationen aus 10-Cent- und 20-Cent-Münzen führen zu einem Gewinn von 1 Euro? Merke dir, wie die Ausgaben vom versteckten Muster a abhängen. Dieses Muster ist der Schlüssel zum Verständnis des Funktionsverhaltens.

Aufgabe 2:

Nun betrachten wir das eigentliche Problem: Bestimme die versteckte Bitfolge **a**=(a_1,a_2,a_3) einer Funktion $f_a(x_1,x_2,x_3)$ aus der folgenden Tabelle.

x_1	x_2	x_3	$f_a(x_1,x_2,x_3)$
0	0	0	0
1	0	0	1
0	1	0	0
0	0	1	1
1	1	0	1
1	0	1	0
0	1	1	1
1	1	1	0

Überlege dir die effizienteste Strategie, um a=(a_1,a_2,a_3) zu bestimmen:

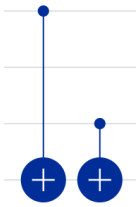
- Wie viele Abfragen benötigst du?
- Welche Eingabewerte (x_1,x_2,x_3) solltest du nutzen, um a=(a_1,a_2,a_3) am effizientesten zu bestimmen?
- Wie lautet die geheime Binärzahl a?

Simulation mit dem IBM Quantum Composer

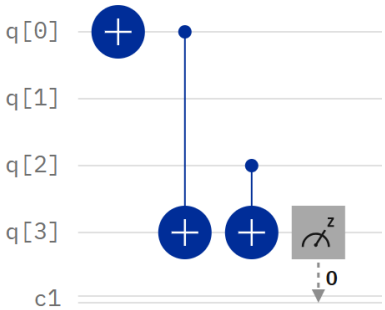
1. Erster Ansatz: Klassische Lösung zum Finden der geheimen Zeichenkette

Öffne die Seite https://quantum.ibm.com/composer/files/new

Die folgende Schaltung implementiert eine bestimmte Bitfolge, die wir herausfinden möchten.

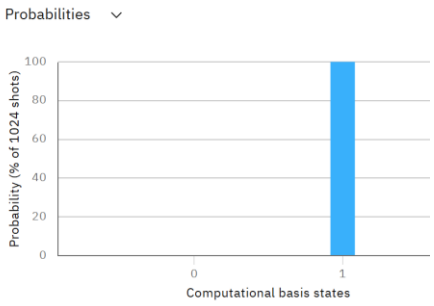


Beispiel: Eingabe x_1=(1,0,0).



- Die Eingabewerte x_1,x_2,x_3 entsprechen den Werten der drei ersten Qubits q[0],q[1],q[2].
- Das Ergebnis der Funktion ist der output von Qubit q[3].
- Alle Qubits haben den Startwert 0. Um von 0 zu 1 zu wechseln, wende ⊕ auf das entsprechende Qubit an.
- Die Messung  des letzten Qubits q[3] ergibt das Ergebnis f(x_1,x_2,x_3).

Der gemessene Wert (in diesem Fall 1) kann im Wahrscheinlichkeitsdiagramm der Simulation abgelesen werden:



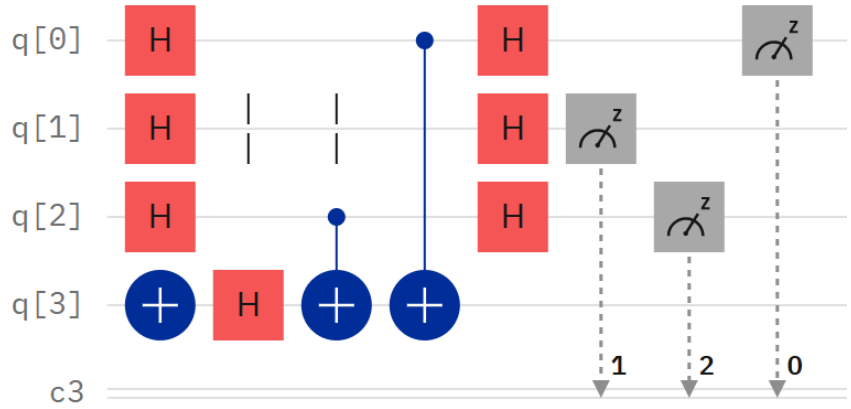
Aufgabe 3: Baue die obige Schaltung nach und verwende den Quantum Composer, um die Tabelle zu vervollständigen:

x_1,x_2,x_2	0,0,0	1,0,0	0,1,0	0,0,1	1,1,0	1,0,1	0,1,1	1,1,1
$f(x_1,x_2,x_2)$		1						

Bestimme die geheime Zeichenkette aus deiner Tabelle.

2. Der Bernstein-Vazirani-Algorithmus

Der Bernstein-Vazirani-Algorithmus ist ein Quanten-Algorithmus, der den untenstehenden Schaltkreis verwendet. Die Funktion wird, wie im klassischen Fall, mit zwei CNOT-Gates implementiert. Die versteckte Bitfolge erhält man nach der Messung der ersten drei Qubits. Wie beim **Deutsch-Algorithmus** erzeugen die anfänglichen **Hadamard-Gates** eine Superposition aller möglichen Eingabewerte.



Baue den Schaltkreis im Quantum Composer nach und bestätige:

Die Simulation liefert dieselbe Bitfolge wie in Aufgabe 3.

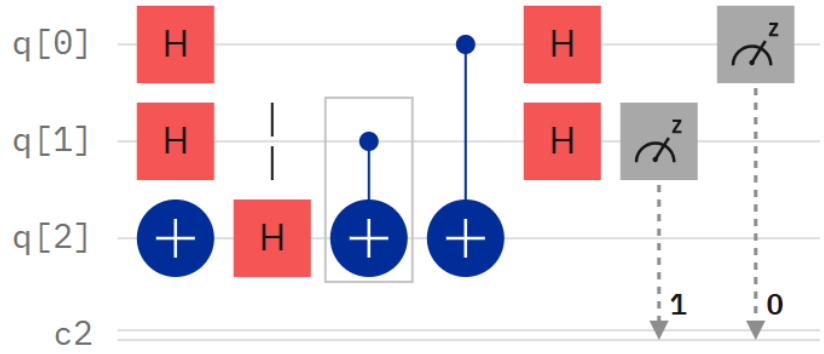
Beachte: Klassisch waren drei Abfragen notwendig, während der Quanten-Algorithmus nur eine Abfrage benötigt.

Um besser zu verstehen, wie der Quanten-Algorithmus das Problem löst, betrachten wir ein Beispiel mit nur zwei Bits $f(x_1,x_2)$. Wir können die Zustandsentwicklung dann einfach mit einer Tabellenkalkulation nachvollziehen.

Aufgabe 4:

Baue die folgende Schaltung **Gate für Gate** nach und ergänze parallel die Tabelle in einem Tabellenkalkulationsprogramm Schritt für Schritt.

Überprüfe nach jedem Schritt: Stimmt deine Berechnung in der Tabelle mit der Simulation im Composer überein?



Hinweise zur Hilfe

- Verwende die Ket-Notation, wobei in der Tabelle der Normierungsfaktor weggelassen wird.
- Zunächst sind alle Qubits in Zustand $|0\rangle$: $|q_0\rangle|q_1\rangle|q_2\rangle \equiv |q_0, q_1, q_2\rangle = |0,0,0\rangle$.
- Das $\oplus$-Gate wird auf $|q_2\rangle$ ausgeführt und ändert den Gesamtzustand zu $|0,0,1\rangle$, was in der Tabelle folgendermaßen notiert wird:

q0		0
q1		0
q2	**x**	1

- Das Hadamard auf $|q_0\rangle$ erzeugt eine Superposition $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. Der Gesamtzustand ist daher $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)|0\rangle|1\rangle = \frac{1}{\sqrt{2}}(|0,0,1\rangle + |1,0,1\rangle)$. Ohne den Normalisierungsfaktor $\frac{1}{\sqrt{2}}$ ergibt sich in der Tabelle dann:

q0		0	**H**	0	1
q1		0		0	0
q2	**x**	1	**x**	1	1

Beantworte anschließend folgende Fragen:

a) Welche Bedeutung haben die beiden Hadamard-Gates am Anfang?

b) Warum befindet sich am Anfang ein $\oplus$-Gate auf dem Ausgabe-Qubit q[2]? Was würde passieren, wenn dieses Gate fehlen würde?

c) Welche Funktion haben die letzten beiden Hadamard-Gates?

d) Welche Bitfolge (a_1,a_2) ergibt sich aus der Messung?