# Průvodce implementace v IBM Qiskit s úvodem do kvantového počítání

## Best Practice Guide

Jiří Tomčala, Marek Lampart

23. září 2022



VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

# Průvodce implementace v IBM Qiskit s úvodem do kvantového počítání

Jiří Tomčala a Marek Lampart

## Obsah

# Předmluva

Kvantové počítače jsou dnes jedním z nejžhavějších technologických odvětví budoucnosti. Tato technologie umožňuje řešit výpočetní problémy, které byly dříve považovány za neřešitelné, a významně ovlivnila kryptografii, chemii, optimalizaci, strojové učení a mnohé další.

Tyto kvantové počítače sice s největší pravděpodobností zcela nenahradí klasické počítače, ale kvantová technologie docela určitě významně změní způsob fungování světa.

Výzkum, který probíhá v mnoha významných technologických společnostech a start-upech, mezi nimiž jsou Google, IBM, Microsoft, Intel, Honeywell a další, vedl k řadě technologických průlomů v budování „bránových" kvantových počítačových systémů. Společnost D-Wave zaujala výrazně odlišný přístup k této nové technologii a věnuje se tzv. „kvantovému žíhání". Mnoho subjektů se zaměřuje na vývoj hardwaru pro kvantové výpočty, protože právě jeho nedokonalost je dnes hlavní překážkou k praktickému využití kvantové technologie. Opět mezi ně patří jak výše uvedení technologičtí giganti, tak relativně nové startupy jako jsou IQM, AQT, QCWare, Rigetti, IonQ, Quantum Circuits a další.

Cílem tohoto textu je seznámit čtenáře se základními principy kvantového počítání v praxi. Naleznete zde základní pojmy a vlastnosti nutné ke konstrukci důležitých algoritmů s návodem implementace v prostředí Qiskit od IBM.

V tomto textu nenajdete úvody do teorie kvantové mechaniky, hluboké matematické struktury, rozsáhlé technické kapitoly o korekcích kvantových chyb, technických problémech implemetace, transpilerech a dalších. Tato a mnohá další témata lze najít například v [1]. Vhodnou alternativou tohoto textu může být [2], který nám byl častou inspirací.

Jako vhodnou prerekvizitu tento text požaduje základní znalosti lineární a tenzorové algebry, základní znalosti programovacího jazyka Python a hlavně chuť a zvědavost skloubenou s touhou proniknout do tajů kvantového počítání či informatiky.

v Ostravě 23. září 2022

autoři
Ing. Jiří Tomčala, Ph.D.
a
prof. RNDr. Marek Lampart, Ph.D.

# 1 Úvod

Ideu kvantového počítače představil v roce 1982 laureát Nobelovy ceny Richard Feynman [3], který poukázal na to, že přesně a efektivně simulovat kvantové mechanické systémy na klasickém počítači není možné, ale že je k tomu zapotřebí nový druh stroje. Počítač, který je sám „postavený z kvantově mechanických prvků, které se řídí zákony kvantové mechaniky". Takovýto systém není možné ze své podstaty simulovat klasickým počítačem. Kvantové počítače, oproti klasickým, využívají jedinečné vlastnosti kvantových systémů (provázání, dekoherence, superpozice, kvantový paralelismus atd.), což jim umožňuje zpracovávat exponenciálně velké množství informací v polynomiálním čase.

V roce 1994 jako první sestavil Peter Shor průlomový kvantový algoritmus [4], který dal jednoznačnou odpověď na palčivou otázku, zda kvantové počítání přináší, ve srovnání s klasickým, podstatné výpočetní benefity. Tento kvantový algoritmus umožňuje faktorizaci čísla v polynomiálním čase, což poskytuje exponenciální zrychlení oproti nejrychlejším klasickým algoritmům. V kryptografii je běžně používaný algoritmus RSA, jehož bezpečnost závisí na skutečnosti, že neexistuje žádný známý účinný klasický algoritmus, který by rozložil dostatečně velké číslo (tzv. veřejný klíč) na prvočísla v reálném čase [5]. Bylo spočítáno, že pokud bychom měli k dispozici kvantový počítač s 4099 perfektně stabilními kvantovými bity, pak by šel tímto kvantovým algoritmem prolomit standardní 2048 bitový RSA klíč za 10 sekund [6]. Připomeňme, že takovýto klíč by šlo prolomit na současném běžném domácím počítači zhruba za 300 biliónů let [6] a na jednom z největších superpočítačů současnosti (Summit, IBM) teoreticky za 90 miliónů let.

Shorův kvantový algoritmus patří mezi první, není však jediný. V současné době je známo a dokonce kategorizováno velké množství kvantových algoritmů [7]. V tomto průvodci osvědčenými postupy se budeme detailně zabývat základními kvantovými algoritmy autorů E. Bernsteina, U. Vaziraniho, L. Grovera a P. Shora. Dříve, než nastíníme jejich principy a ukážeme návod implementace ve volně dostupném vývojovém prostředí Qiskit od IBM [8], formulujme nejprve základní aparát nutný ke konstrukci a pochopení zmíněných algoritmů.

# 2 Qubit

V klasické počítačové vědě se používá základní jednotka informace *bit*. Jedná se o abstraktní pojem. Hodnota *klasického bitu* je 0 nebo 1.

Na rozdíl od počítače klasického, počítač kvantový pracuje se stavy, které vychází ze světa kvantové fyziky [9, 10], což je důvodem, proč je kvantové počítání diametrálně odlišné od toho klasického. V případě kvantového počítání se objevují dvě neobvyklé vlastnosti, které se v klasickém případě nevyskytují, a to superpozice a provázanost. První vlastnost *superpozice* vychází z toho, že se kvantová částice ve stejné chvíli může nacházet ve stavech, které se navzájem vylučují, například nachází se „tady" i „tam" současně. Jednotky kvantové informace - kvantové bity - tedy nejsou jen 0 či 1, ale taky jejich libovolná kombinace, superpozice těchto stavů. Druhá vlastnost je příčinou extrémně vysoké efektivity kvantového počítání. *Provázanost* (entanglement) je nemožnost matematického oddělení stavů dvou různých fyzikálních objektů, například elementů kvantového počítače [11].

*Kvantový bit* (zkráceně *qubit* [12]) je dvoudimenzionální kvantově mechanický systém, který je ve stavu

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,, \tag{1}$$

kde $\alpha$ i $\beta$ jsou komplexní čísla, což jsou *amplitudy* kvantových stavů $|0\rangle$ resp. $|1\rangle$. Druhé mocniny absolutních hodnot těchto amplitud představují pravděpodobnosti, že daný kvantový

stav bude pozorován v okamžiku měření. Pro amplitudy proto platí $|\alpha|^2 + |\beta|^2 = 1$.

V této definici se užívá standardní *bra-ketová*[1] (Diracova[2]) notace, tady označuje

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \text{ a } |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \tag{3}$$

nejpoužívanější bázi v kvantovém počítání, takzvanou *výpočetní bázi*. Mimo výše uvedenou bázi je možno použít libovolnou ortonormální bázi, například

$$|+\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \text{ a } |-\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix}. \tag{4}$$

Geometrická interpretace kvantového stavu qubitu je možná pomocí takzvané *Blochovy sféry* zobrazené na obr. 1. Tedy qubit je možno reprezentovat pomocí dvou úhlů $\phi$ a $\theta$. Přesněji, všechny lineární kombinace $\alpha\,|0\rangle + \beta\,|1\rangle$ v $\mathbb{C}^2$ odpovídají bodům $(\phi, \theta)$ na jednotkové (Blochově) sféře. V tomto případě $\alpha = \cos(\theta/2)$ a $\beta = e^{i\phi}\sin(\theta/2)$:

$$\begin{aligned} |\psi\rangle &= \cos(\theta/2)\,|0\rangle + e^{i\phi}\sin(\theta/2)\,|1\rangle & (5)\\ &= \cos(\theta/2)\,|0\rangle + (\cos(\phi) + i\sin(\phi))\sin(\theta/2)\,|1\rangle\,, & (6) \end{aligned}$$

kde $0 \leq \theta \leq \pi$ a $0 \leq \phi < 2\pi$.

# 3 Kvantové registry

Kvantové výpočty, stejně jako ty klasické, neprobíhají na jediném qubitu, ale na *kvantovém registru*, který zahrnuje více qubitů najednou. Kvantový registr je tedy analogie klasického procesorového registru a kvantové počítače provádějí výpočty pomocí manipulací qubitů v rámci daného registru. Každý qubit v registru je superpozicí prvků výpočetní báze $|0\rangle$ a $|1\rangle$. Tedy registr s $n$ qubity je superpozicí všech možných $2^n$ bitových řetězců, které mohu být reprezentovány $n$ bity. Stavový prostor $n$ kvantového registru je lineární kombinací $n$ bázových vektorů délky $2^n$

$$|\psi_n\rangle = \sum_{i=0}^{2^n-1} \alpha_i\,|i\rangle \tag{7}$$

kde $i$ je celé číslo v desítkové soustavě reprezentující číslo délky $n$ ve dvojkové soustavě.

Například tří qubitový registr má tvar

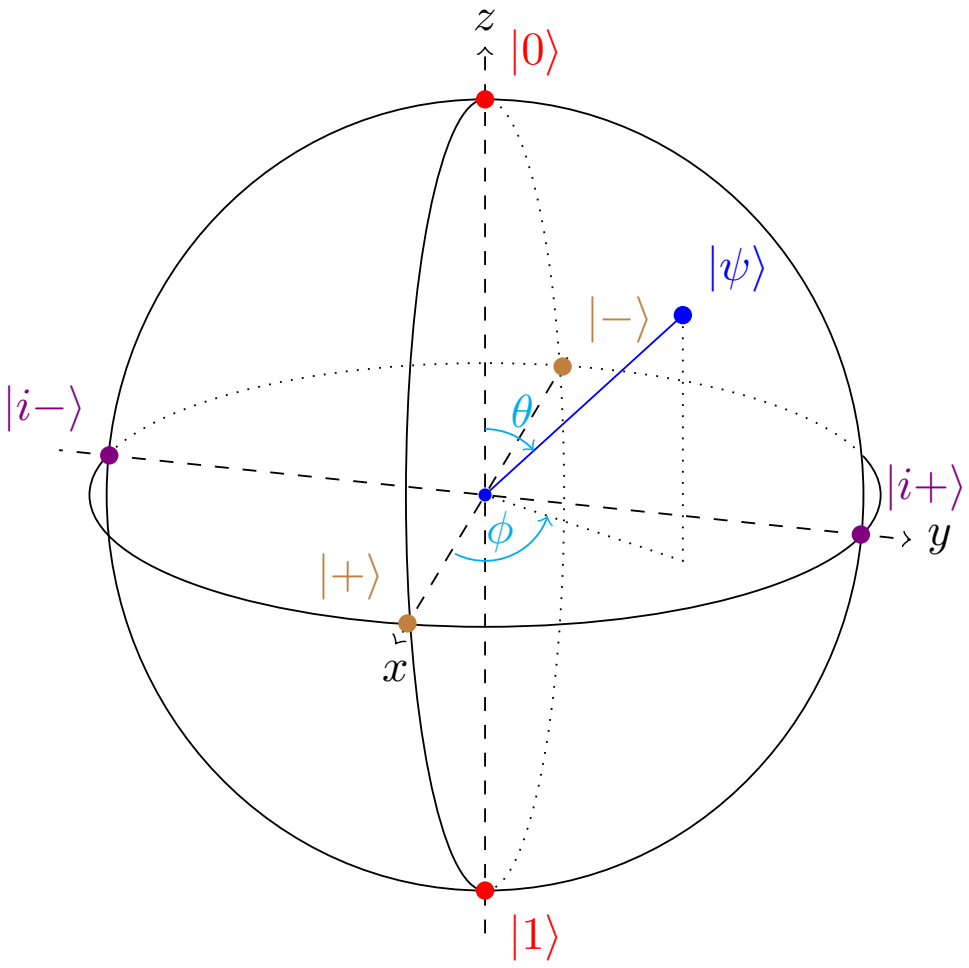
$$|\psi_3\rangle = \alpha_0\,|000\rangle + \alpha_1\,|001\rangle + \alpha_2\,|010\rangle + \alpha_3\,|011\rangle + \alpha_4\,|100\rangle + \alpha_5\,|101\rangle + \alpha_6\,|110\rangle + \alpha_7\,|111\rangle \tag{8}$$

[1] V této notaci $|u\rangle$ označuje sloupcový vektor

$$|u\rangle = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix} \tag{2}$$

nazývaný *ket-u*. Duální vektor $\langle u| = \overline{u^T} = (\overline{u}_1\,\overline{u}_2\,\ldots\,\overline{u}_n)$ je *bra-u*, zde $\overline{u}$ je komplexně sdružený vektor vektoru $u$.

[2] Jedná se o způsob zápisu vektorů, který je běžně používaný v kvantové mechanice a kvantové teorii pole. Jde o zápis vektorů v Hilbertově prostoru, který zavedl P.A.M. Dirac.




Obrázek 1: Blochova sféra

Každá bitová konfigurace v kvantové superpozici je označena jako tenzorový součin jednotlivých qubitů[3]. Například $|010\rangle$, která reprezentuje v bitovém řetězci číslo 2, má tvar:

$$|010\rangle = |0\rangle \otimes |1\rangle \otimes |0\rangle = \begin{pmatrix}1\\0\end{pmatrix} \otimes \begin{pmatrix}0\\1\end{pmatrix} \otimes \begin{pmatrix}1\\0\end{pmatrix} = (0\,0\,1\,0\,0\,0\,0\,0)^{\mathrm{T}}. \tag{10}$$

Pro (7) platí normalizační podmínka pro jednotlivé amplitudy pravděpodobností:

$$\sum_{i=0}^{2^n-1} |\alpha_i|^2 = 1. \tag{11}$$

Logicky, součet pravděpodobností všech možných stavů musí být 1, protože žádný jiný stav nastat nemůže a zároveň se jednotlivé stavy navzájem vylučují. Celkově tedy, kvantové registry jsou přímým rozšířením qubitů.

# 4 Kvantové logické brány

Klasické logické brány jsou matematicky popsány pomocí Booleovské algebry. Kvantové logické brány fungují na podobném principu. Kvantové logické brány aplikované na kvantové registry zobrazují kvantovou superpozici na jinou a společně umožňují vývoj systému do nějakého požadovaného konečného stavu, správné odpovědi.

Kvantové logické brány jsou matematicky reprezentovány pomocí transformací matic (lineárních operací) aplikovaných na kvantový registr tenzorováním transformačních matic maticí reprezentující daný registr. Všechny matice odpovídající kvantové logické bráně jsou *unitární*[4]. Unitární transformace prováděné na jednom qubitu mohou být efektivně vizualizovány na Blochově sféře.

Tak jako v případě klasických logických bran je i v případě kvantových logických bran standardní množinou používaných bran množina zavedená v [13].

[3] Připomeňme, že $\otimes$ značí operaci *tenzorového součinu* definovaného pro dva vektory $|u\rangle$ a $|v\rangle$ dimenze $m$ resp. $n$ následujícím způsobem

$$|u\rangle\,|v\rangle = |uv\rangle = |u\rangle \otimes |v\rangle = \begin{pmatrix}u_1\\u_2\\\vdots\\u_m\end{pmatrix} \otimes \begin{pmatrix}v_1\\v_2\\\vdots\\v_n\end{pmatrix} = \begin{pmatrix}u_1v_1\\u_1v_2\\\vdots\\u_1v_n\\u_2v_1\\u_2v_2\\\vdots\\u_2v_n\\\vdots\\u_mv_1\\u_mv_2\\\vdots\\u_mv_n\end{pmatrix}. \tag{9}$$

Výsledný vektor $|uv\rangle$ je tedy dimenze $mn$. Tenzorový součin je distributivní i asociativní, ale obecně není komutativní.

[4] Komplexní matice $U$ je *unitární* právě tehdy, když $U^{-1} = U^\dagger$, kde $U^\dagger$ je matice konjugovaná k matici $U$ ($U^\dagger = \overline{U}^T$). Navíc platí $UU^\dagger = U^\dagger U = I$.

Nejprve uveďme *Hadamardův operátor* $H$

$$-\boxed{H}- = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix} = \frac{|0\rangle + |1\rangle}{\sqrt{2}}\langle 0| + \frac{|0\rangle - |1\rangle}{\sqrt{2}}\langle 1|$$

Aplikováním tohoto operátoru na $|0\rangle$ nebo $|1\rangle$ dostaneme superpozici stavů $|0\rangle$ a $|1\rangle$ se stejnou pravděpodobností jejich výskytu. Geometricky, zobrazeno na Blochově sféře, Hadamardův operátor provede nejprve rotaci $\pi/2$ kolem osy $y$ a pak rotaci $\pi$ kolem osy $x$.

Dalším příkladem jsou *Pauliho brány* $X$, $Y$ a $Z$, které odpovídají rotaci o úhel $\pi$ kolem osy $x$, $y$ či $z$ respektive. K zavedení těchto bran je zapotřebí *vnitřního* a *vnějšího součinu.* [5]

Pauliho brána $X$ přehodí amplitudy $|0\rangle$ a $|1\rangle$

$$-\boxed{X}- = \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix} = |1\rangle\langle 0| + |0\rangle\langle 1|$$

Pauliho brána $Y$ přehodí amplitudy $|0\rangle$ a $|1\rangle$, násobí je komplexní jednotkou $i$ a neguje amplitudu $|1\rangle$

$$-\boxed{Y}- = \begin{bmatrix}0 & -i\\ i & 0\end{bmatrix} = i\,|1\rangle\langle 0| - i\,|0\rangle\langle 1|$$

Pauliho brána $Z$ neguje amplitudu $|1\rangle$ a ponechává amplitudu $|0\rangle$ beze změny

$$-\boxed{Z}- = \begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix} = |0\rangle\langle 0| - |1\rangle\langle 1|$$

Brána fázového posunu $R_\varphi$ nemění amplitudu $|0\rangle$, ale mění fázi $|1\rangle$ o faktor $e^{i\varphi}$ pro každou hodnotu $\varphi$

$$-\boxed{R_\varphi}- = \begin{bmatrix}1 & 0\\ 0 & e^{i\varphi}\end{bmatrix} = |0\rangle\langle 0| + e^{i\varphi}\,|1\rangle\langle 1|$$

Poznamenejme, že $Z$ je speciální případ $R_\varphi$ pro $\varphi = \pi$. Dalším speciálním případem $R_\varphi$ pro $\varphi = \pi/2$ je *fázová* brána $S$, která mění fázi $|1\rangle$ faktorem $i$

$$-\boxed{S}- = \begin{bmatrix}1 & 0\\ 0 & i\end{bmatrix} = |0\rangle\langle 0| + i\,|1\rangle\langle 1|$$

Jako poslední případ uveďme bránu $T$, je to brána $R_\varphi$ pro $\varphi = \pi/4$

$$-\boxed{T}- = \begin{bmatrix}1 & 0\\ 0 & e^{i\pi/4}\end{bmatrix} = |0\rangle\langle 0| + e^{i\pi/4}\,|1\rangle\langle 1|$$

[5] Zde definujeme *vnitřní součin* $\langle u|v\rangle$ vektorů $u$ a $v$ téhož prostoru jako součin vektorů $\overline{u}^T$ a $v$:

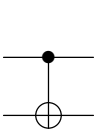
$$\langle u|v\rangle = \overline{u}^T v = (\overline{u}_1\,\overline{u}_2\,\ldots\,\overline{u}_n)\begin{pmatrix}v_1\\ v_2\\ \vdots\\ v_n\end{pmatrix} = \overline{u}_1 v_1 + \overline{u}_2 v_2 + \cdots + \overline{u}_n v_n,$$

*vnější součin* $|v\rangle\langle u|$ vektorů $u$ a $v$ prostorů dimenze $m$ reps. $n$:

$$|v\rangle\langle u| = v\overline{u}^T = \begin{pmatrix}v_1\\ v_2\\ \vdots\\ v_n\end{pmatrix}(\overline{u}_1\,\overline{u}_2\,\ldots\,\overline{u}_m) = \begin{bmatrix}v_1\overline{u}_1 & v_1\overline{u}_2 & \ldots & v_1\overline{u}_m\\ v_2\overline{u}_1 & v_2\overline{u}_2 & \ldots & v_2\overline{u}_m\\ \vdots & \vdots & \ddots & \vdots\\ v_n\overline{u}_1 & v_n\overline{u}_2 & \ldots & v_n\overline{u}_m\end{bmatrix}$$

Výše uvedené brány jsou unární, což znamená, že se vždy působí jedním qubitem na danou bránu. S takovými operacemi si ale nevystačíme, potřebujeme další, binární i ternární. Mezi unární patří NOT, mezi binární AND, OR, CNOT, XOR, SWAP a ternární CCNOT a mnohé další viz [14].

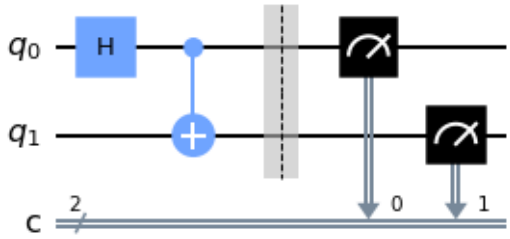
Nejdůležitější a taky nejpoužívanější binární kvantovou branou je CNOT



$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

# 5 Provázanost kvantových stavů

Jak již bylo řečeno, *provázanost* (entanglement) je nemožnost matematického oddělení dvou kvantových stavů. Co konkrétně je tím myšleno a jak se to projevuje lze nejlépe vysvětlit na příkladě jednoduchého kvantového obvodu, zobrazeného na obr. 2.



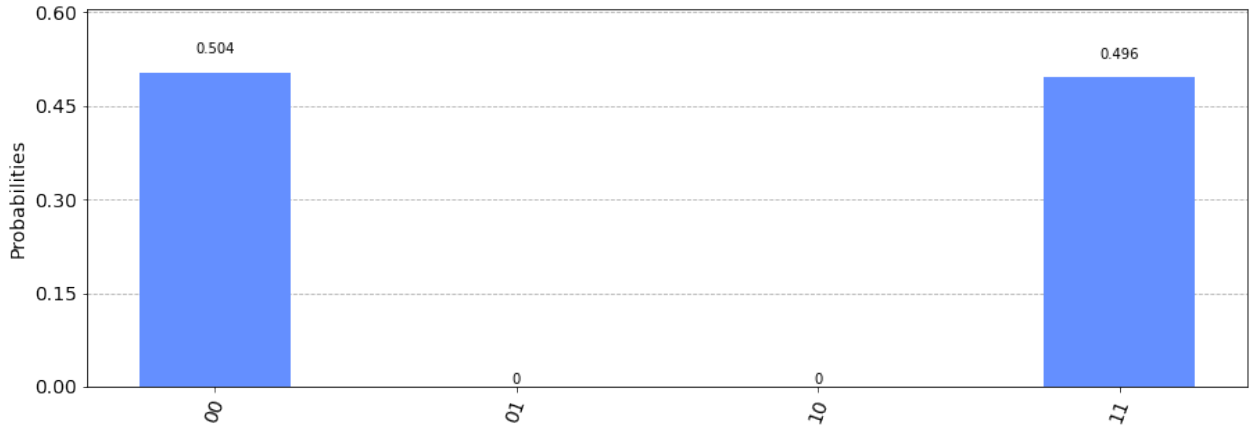

Obrázek 2: Kvantový obvod pro vytvoření provázanosti mezi qubity $q_0$ a $q_1$.

```python
# inicializace
import matplotlib.pyplot as plt

# import Qiskitu
from qiskit import IBMQ, Aer
from qiskit.providers.ibmq import least_busy
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister, transpile, assemble

# import zakladnich vizualizacnich nastroju
from qiskit.visualization import plot_histogram

# Je zapotrebi vytvorit kvantovy obvod s n+m qubity
# Taky bude zapotrebi n klasickych bitu, do kterych se bude
# zapisovat namereny vystup
ko = QuantumCircuit(2, 2)

# Nastaveni qubitu q0 do stavu superpozice |+>
ko.h(0)

# Nastaveni vstupu funkce 6^x mod 35 na |001>
ko.cx(0,1)

# Zacatek casti obvodu, kde dochazi k mereni kvantovych stavu qubitu
ko.barrier()

# Mereni kvantovych stavu qubitu
ko.measure(0,0)
ko.measure(1,1)

ko.draw()  # Zobrazeni
           # obvodu
```

Zdrojový kód 1: Implementace kvantového obvodu pro vytvoření provázanosti mezi qubity $q_0$ a $q_1$ v Qiskitu.

```
# Pouziti simulatoru:
# -------------------
simulator = Aer.get_backend('aer_simulator')
transpilovany_ko = transpile(ko, simulator)
objekt_ko = assemble(transpilovany_ko, shots=1024)
vysledky = aer_sim.run(objekt_ko).result()
pocty_jednotlivych_kombinaci = vysledky.get_counts()

plot_histogram(pocty_jednotlivych_kombinaci) # Zobrazeni
                                             # histogramu
```

Zdrojový kód 2: Spuštění kvantového obvodu na IBM simulátoru.




Obrázek 3: Výsledné rozložení pravděpodobností naměřeného výstupu kvantového obvodu z obr. 2 po spuštění na IBM simulátoru (*ibmq_qasm_simulator*).

Z výsledného rozložení pravděpodobností naměřených na simulátoru (obr. 3) vyplývá, že při měření na výstupu tohoto obvodu nabývají $q_0$ a $q_1$ vždy stejných hodnot. I kdybychom tyto dva qubity od sebe vzdálili na libovolnou vzdálenost, pak v momentě měření jednoho z nich bychom věděli, jaký bude výsledek měření toho druhého qubitu. Je to tím, že jejich kvantové stavy jsou tzv. provázané (entangled). Důležité je ovšem také si uvědomit, že dokud stav jednoho z takto provázaných qubitů nezměříme, je stejná pravděpodobnost, že výsledek obou bude 0 stejně jako, že výsledek obou bude 1. Proč to tak je, lze odvodit matematicky, ale pro potřeby tohoto průvodce je postačující se podívat, co se při spuštění tohoto obvodu děje s kvantovými stavy $q_0$ a $q_1$.

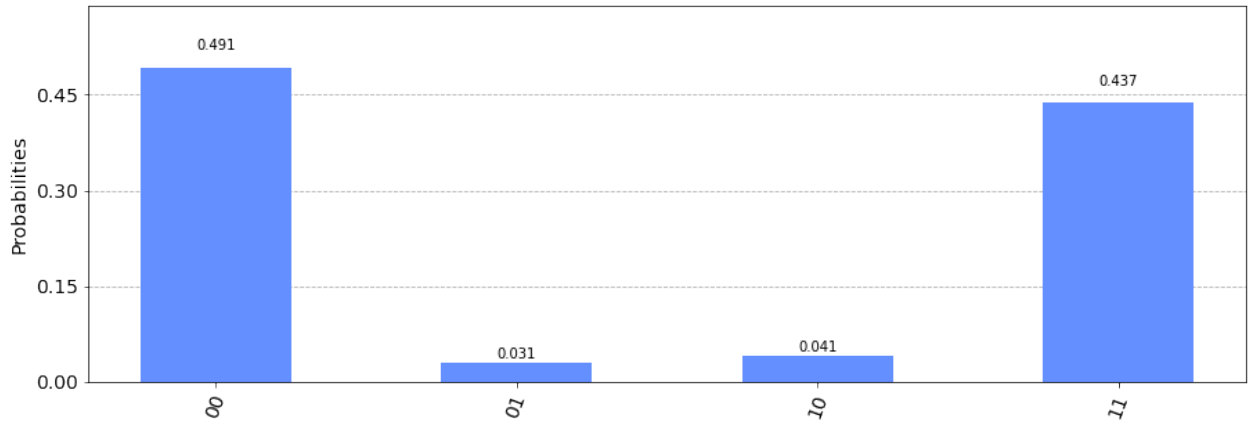
Nejprve se pomocí Hadamardovy brány natočí $q_0$ do stavu superpozice, kdy je stejná pravděpodobnost $|0\rangle$ i $|1\rangle$, a tento stav je pak podmínkou provedení inverze (rotace kolem osy $x$) kvantového stavu $q_1$ pomocí brány CNOT. Jelikož $q_1$ byl na počátku ve stavu $|0\rangle$, pak pokud se superponovaný $q_0$ při provedení CNOT projeví jako $|0\rangle$, nestane se s kvantovým stavem $q_1$ vůbec nic a na výstupu pak naměříme u obou stav 0. Pokud se ovšem superponovaný $q_0$ při provedení CNOT projeví jako $|1\rangle$, dojde k inverzi kvantového stavu $q_1$ a na výstupu pak naměříme u obou stav 1.

```
# Pouziti kvantoveho pocitace IBM:
# --------------------------------
# Nahrani IBMQ uctu a zjisteni nejmene vytizeneho verejne dostupneho
# 5-qubitoveho kvantoveho pocitace IBM
IBMQ.load_account()
poskytovatel = IBMQ.get_provider(hub='ibm-q')
poskytovatel.backends()
kvantovy_pocitac = least_busy(poskytovatel.backends(filters=lambda x: x.configuration().
    n_qubits >= 5 and
                                not x.configuration().simulator and x.status().operational
    ==True))

print("V teto chvili nejmene vytizeny verejne dostupny 5-qubitovy IBM kvantovy pocitac: ",
    kvantovy_pocitac)
```

```
12
13 # Vlozeni pozadavku na spusteni kvantoveho obvodu na vybranem kvantovem
14 # pocitaci a monitorovani jeho spusteni kazde 2 sekundy
15 from qiskit.tools.monitor import job_monitor
16
17 transpilovany_ko = transpile(ko, kvantovy_pocitac)
18 job = kvantovy_pocitac.run(transpilovany_ko, shots=1024)
19
20 job_monitor(job, interval=2)
21
22 # Ziskani vysledku ze spusteni kvantoveho obvodu
23 vysledky = job.result()
24 pocty_jednotlivych_kombinaci = vysledky.get_counts()
25
26 plot_histogram(pocty_jednotlivych_kombinaci) # Zobrazeni
27                                               # histogramu
```

Zdrojový kód 3: Spuštění kvantového obvodu na kvantovém počítači IBM.




Obrázek 4: Výsledné rozložení pravděpodobností naměřeného výstupu kvantového obvodu z obr. 2 po spuštění na kvantovém počítači IBM (*ibmq_quito*).

Výsledné rozložení pravděpodobností naměřených na opravdovém kvantovém počítači (obr. 4) ukazuje na nedokonalost dnešních kvantových počítačů, kdy při provádění operace CNOT se superponovaný $q_0$ projeví jinak, než při jeho pozdějším měření. V histogramu se tento fakt projevil tím, že se objevily další dva možné výsledky s pravděpodobnostmi 3,1 a 4,1 %. Celkově tedy tento triviální obvod na reálném kvantovém počítači selhává přibližně v 7 % případů!

# 6 Bernstein-Vaziraniho algoritmus

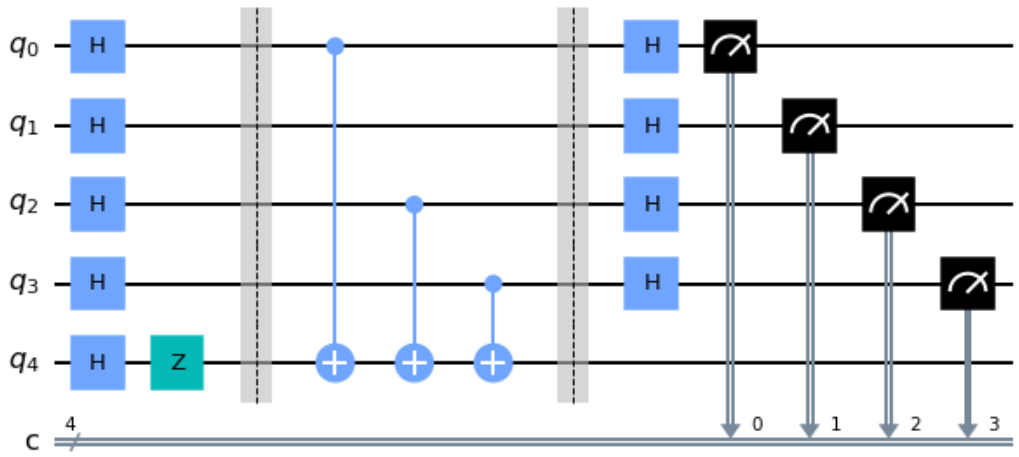
Tento algoritmus je v podstatě variantou Deutsch-Jozsa algoritmu. Jeho implementací je dokonce stejný kvantový obvod.

Zatímco Deutsch-Jozsa algoritmus je určen pro rozhodnutí, zda je nějaká neznámá funkce konstantní nebo balancovaná, cílem Bernstein-Vaziraniho algoritmu je odhalit tajný kód v této neznámé funkci nastavený.

Tato neznámá funkce je implementovaná v tzv. černé skříňce (black box), jejíž vnitřní zapojení není známé. Taková černá skříňka se v odborné literatuře označuje taky někdy jako *orákulum (oracle)*. Ve schématech kvantových obvodů se ovšem toto orákulum označuje jako $U_f$.

```
1 # inicializace
2 import matplotlib.pyplot as plt
3 import numpy as np
```



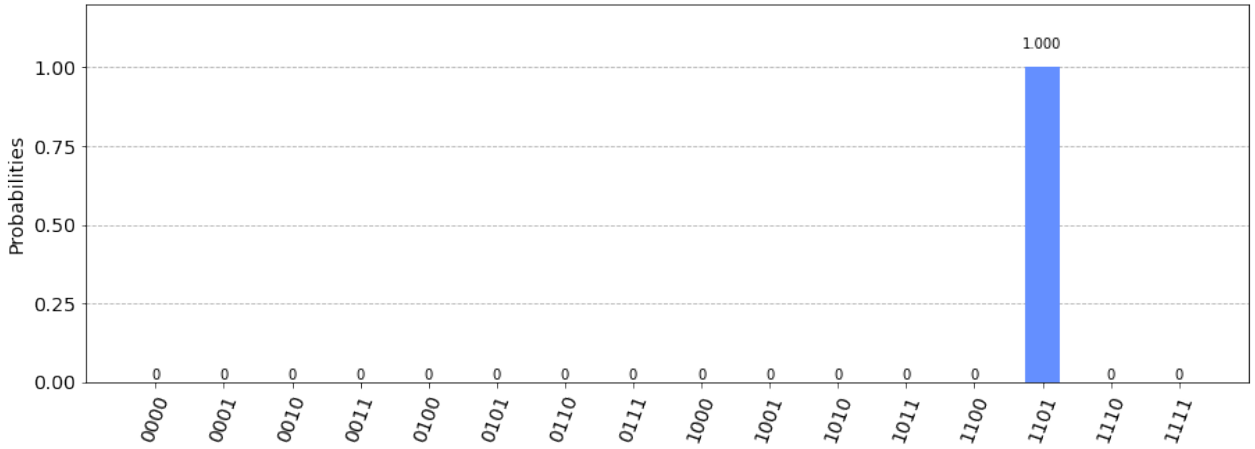

Obrázek 5: Kvantový obvod implementující Bernstein-Vaziraniho algoritmus. Tajný kód je 1101.

```
4
5  # import Qiskitu
6  from qiskit import IBMQ, Aer
7  from qiskit.providers.ibmq import least_busy
8  from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister, transpile, assemble
9
10 # import zakladnich vizualizacnich nastroju
11 from qiskit.visualization import plot_histogram
12
13 n = 4 # pocet qubitu tajneho kodu
14 tajnyKod = '1101'
15
16 # Je zapotrebi vytvorit kvantovy obvod s n qubity, plus jeden pomocny qubit
17 # Taky bude zapotrebi n klasickych bitu, do kterych se bude zapisovat
18 # namereny vystup
19 ko = QuantumCircuit(n+1, n)
20
21 # Nastaveni pomocneho qubitu do stavu |->
22 ko.h(n)
23 ko.z(n)
24
25 # Nastaveni vsech vstupu Orakula do stavu |+>
26 for i in range(n):
27     ko.h(i)
28
29 # Zacatek Orakula - tato bariera nijak neovlivnuje stav qubitu,
30 # slouzi zde pouze k oznaceni mista v obvodu, kde zacina Orakulum
31 ko.barrier()
32
33 # Orakulum
34 tajnyKod = tajnyKod[::-1] # Obraceni poradi, aby odpovidalo usporadani
35                           # qubitu v Qiskitu
36 for q in range(n):
37     if tajnyKod[q] == '1':
38         ko.cx(q, n)
39
40 # Konec Orakula - tato bariera nijak neovlivnuje stav qubitu,
41 # slouzi zde pouze k oznaceni mista v obvodu, kde konci Orakulum
42 ko.barrier()
43
44 # Pouziti Hadamardovych bran na vystup z Orakula, cimz dojde k odhaleni
45 # tajneho kodu
46 for i in range(n):
47     ko.h(i)
48
49 # Mereni vystupu obvodu
50 for i in range(n):
51     ko.measure(i, i)
52
53 ko.draw() # Zobrazeni
```

```
54          # obvodu
```

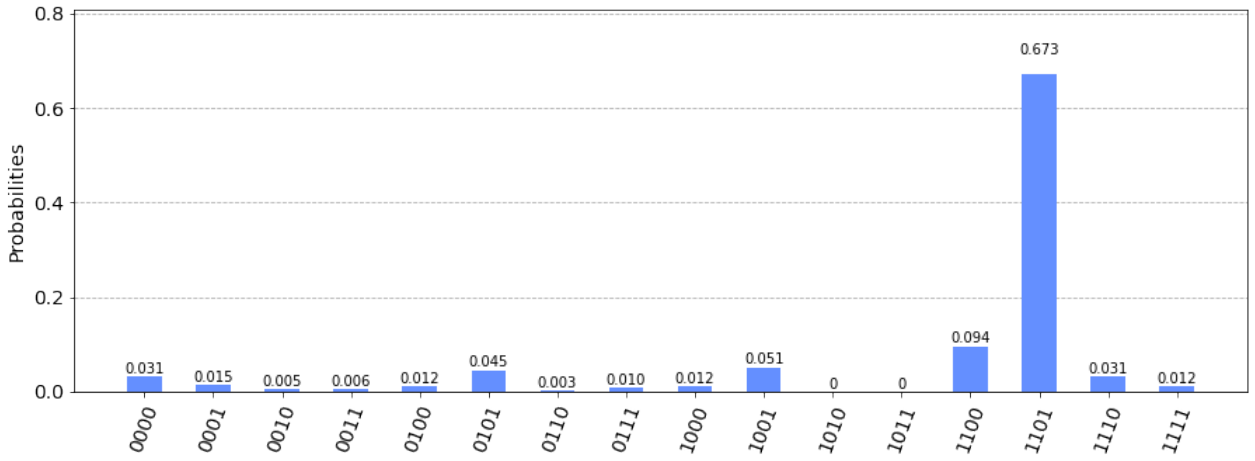
Zdrojový kód 4: Implementace Bernstein-Vaziraniho algoritmu v Qiskitu.

Na simulátoru pak lze tento obvod spustit opět pomocí stejného zdrojového kódu 2. Výsledek je zobrazen v obr. 6.




Obrázek 6: Výsledné rozložení pravděpodobností naměřeného výstupu kvantového obvodu z obr. 5 po spuštění na IBM simulátoru (*ibmq_qasm_simulator*).

Na kvantovém počítači pak lze tento obvod spustit opět pomocí stejného zdrojového kódu 3. Výsledek je zobrazen v obr. 7.




Obrázek 7: Výsledné rozložení pravděpodobností naměřeného výstupu kvantového obvodu z obr. 5 po spuštění na kvantovém počítači IBM (*ibmq_quito*).

# 7 Groverův prohledávací algoritmus

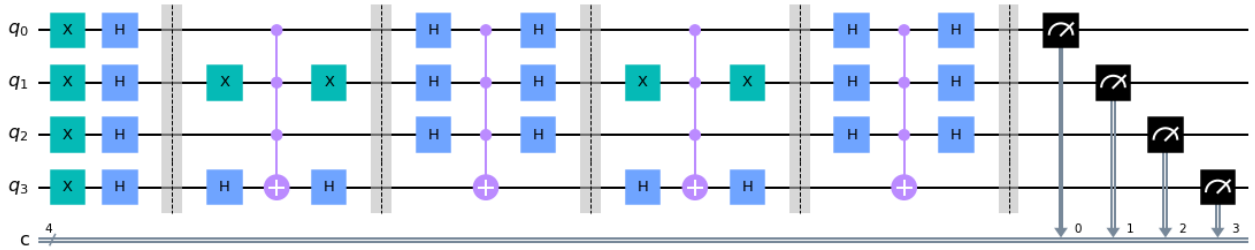
Tento algoritmus je vhodný pro vyhledávání v tzv. nestrukturovaných datech. Při takovém vyhledávání může Groverův algoritmus dosáhnout až kvadratického zrychlení. Je to díky tzv. triku se zesílením amplitudy.

O co se jedná? Velmi stručně řečeno je zde opět zapotřebí sestavit orákulum, které pro hledanou vstupní kombinaci otočí fázi výstupního kvantového stavu. Za ním pak následuje

tzv. *difuzér*, který zesílí amplitudu kvantového stavu s obrácenou fází na úkor amplitud ostatních kvantových stavů a zároveň tomuto zesílenému kvantovému stavu s obrácenou fází opět tuto fázi otočí. Na výstupu difuzéru je tak u hledané kombinace vyšší pravděpodobnost než u ostatních možných vstupních kombinací.

Tento postup (orákulum a difuzér) lze opět zopakovat, čímž dojde k ještě většímu zvýšení pravděpodobnosti výskytu hledané kombinace na výstupu a zároveň snížení pravděpodobností ostatních možných kombinací. Obecně je zapotřebí provést přibližně $\sqrt{N}$ průchodů těmito dvěma obvody (orákulum a difuzér), aby byla dosažena maximální možná pravděpodobnost hledané kombinace, kde $N$ je počet všech možných vstupních kombinací.

Při klasickém hledání v nestrukturovaných datech je zapotřebí až $N - 1$ pokusů, aby byla hledaná vstupní kombinace nalezena, zatímco Groverův algoritmus k tomu samému potřebuje maximálně $\sqrt{N}$ pokusů. Proto tento algoritmus může dosáhnout kvadratického zrychlení a v některých případech dokonce i vyššího, protože již po několika průchodech orákulem a difuzérem může být jasné, u které kombinace se pravděpodobnost postupně zvyšuje.



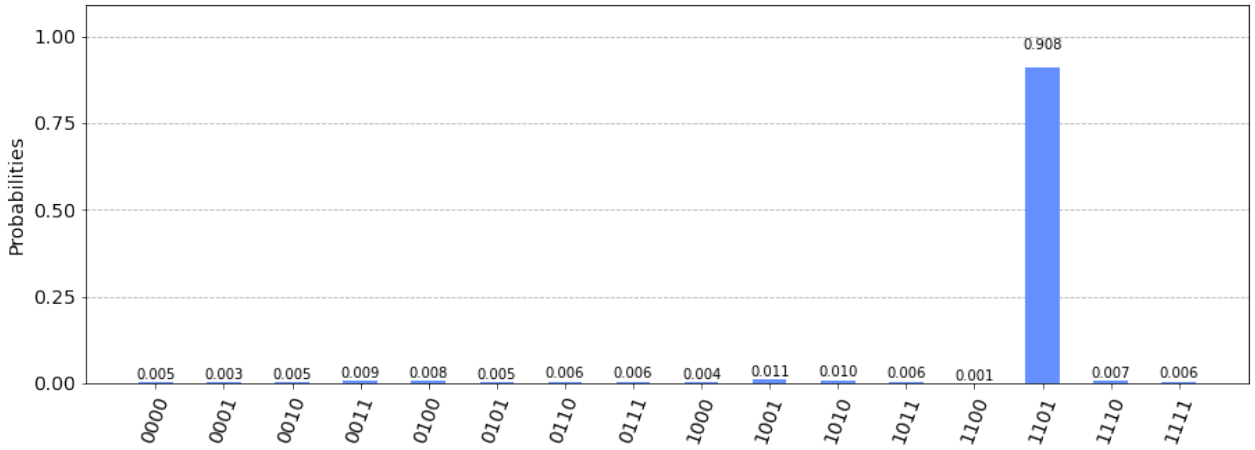

Obrázek 8: Kvantový obvod implementující Groverův algoritmus. Hledaná vstupní kombinace je 1101.

```
# inicializace
import matplotlib.pyplot as plt
import numpy as np

# import Qiskitu
from qiskit import IBMQ, Aer
from qiskit.providers.ibmq import least_busy
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister, transpile, assemble
from qiskit.circuit.library import C3XGate

# import zakladnich vizualizacnich nastroju
from qiskit.visualization import plot_histogram

n = 4 # pocet qubitu tajneho kodu
hledanaVstupniKombinace = '1101'
hledanaVstupniKombinace = hledanaVstupniKombinace[::-1] # Obraceni poradi, aby odpovidalo
    usporadani qubitu v Qiskitu

# Je zapotrebi vytvorit kvantovy obvod s n qubity
# Taky bude zapotrebi n klasickych bitu, do kterych se bude zapisovat
# namereny vystup
ko = QuantumCircuit(n, n)

# Nastaveni vsech vstupu orakula do stavu |->
for i in range(n):
    ko.x(i)
    ko.h(i)

for j in range(2):  # Pro nejlepsi vysledek je zapotrebi dvakrat provest
                    # pruchod orakulem a difuzerem

    # Zacatek orakula - tato bariera nijak neovlivnuje stav qubitu.
    # Slouzi zde pouze k oznaceni mista v obvodu, kde zacina orakulum.
    ko.barrier()

    # Orakulum
    for i in range(n):
```

```
        if hledanaVstupniKombinace[i] == '0':
            ko.x(i)
    ko.h(n-1)
    ko.append(C3XGate(),[0,1,2,3])
    ko.h(n-1)
    for i in range(n):
        if hledanaVstupniKombinace[i] == '0':
            ko.x(i)

    # Konec orakula
    ko.barrier()

    # Difuzer
    for i in range(n-1):
        ko.h(i)
    ko.append(C3XGate(),[0,1,2,3])
    for i in range(n-1):
        ko.h(i)


# Zacatek mereni kvantoveho stavu qubitu
ko.barrier()

# Mereni kvantoveho stavu qubitu
for i in range(n):
    ko.measure(i, i)

ko.draw()  # Zobrazeni
           # obvodu
```

Zdrojový kód 5: Implementace Groverova algoritmu v Qiskitu.

Na simulátoru pak lze tento obvod spustit pomocí stejného zdrojového kódu 2. Výsledek je zobrazen v obr. 9.



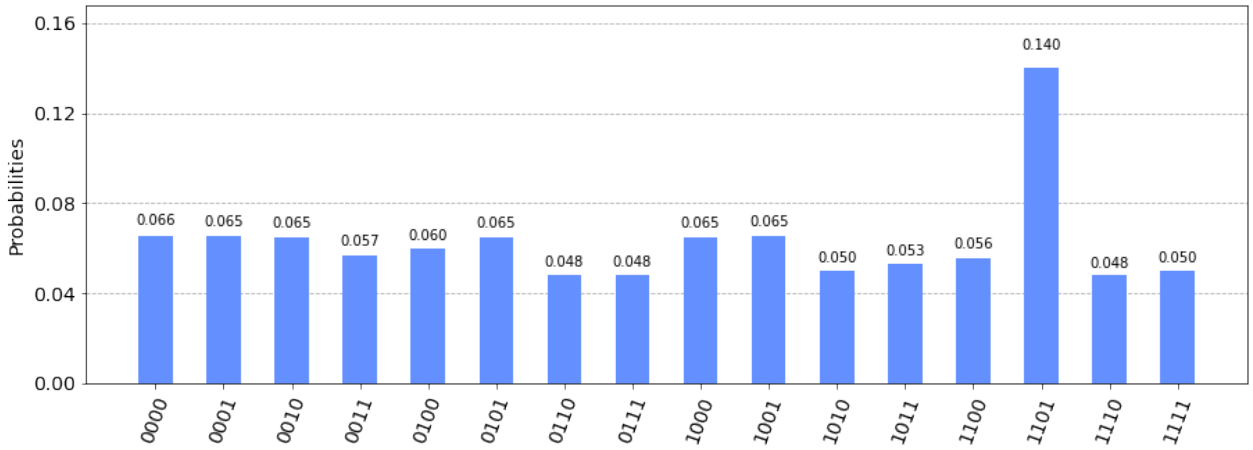

Obrázek 9: Výsledné rozložení pravděpodobností naměřeného výstupu kvantového obvodu z obr. 8 po spuštění na IBM simulátoru (*ibmq_qasm_simulator*).

Na kvantovém počítači pak lze tento obvod spustit pomocí stejného zdrojového kódu 3. Výsledek je zobrazen v obr. 10.

# 8 Shorův faktorizační algoritmus

Shorův algoritmus [4] lze použít pro faktorizaci celých čísel v polynomiálním čase. Vzhledem k tomu, že nejlepší klasické algoritmy vyžadují pro tuto faktorizaci superpolynomiální čas, nejrozšířenější kryptovací systém (RSA) je založen na tom, že faktorizace dostatečně velkých celých čísel je v podstatě nemožná. V současnosti je doporučovaným standardem použití RSA



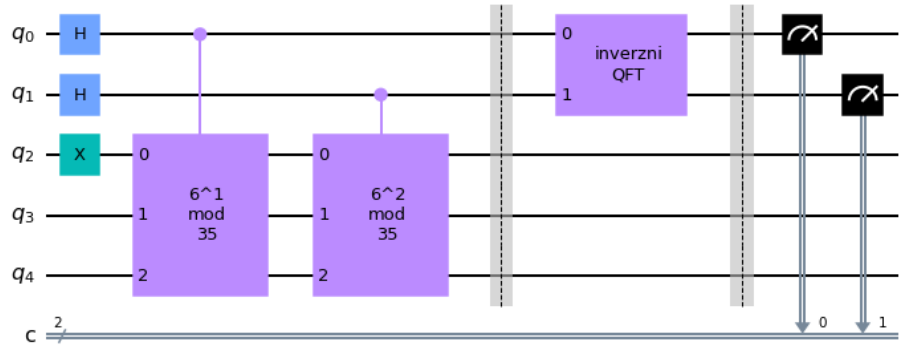

Obrázek 10: Výsledné rozložení pravděpodobností naměřeného výstupu kvantového obvodu z obr. 8 po spuštění na kvantovém počítači IBM (*ibmq_quito*).

s šířkou šifrovacího klíče 2048 bitů, jehož faktorizace by na dnešním běžném PC trvalo přibližně $3 \times 10^{14}$ let. Na jednom z největších superpočítačů současnosti (Summit, IBM) by pak taková faktorizace trvala teoreticky 90 miliónů let.

Podrobné vysvětlení principu Shorova algoritmu je nad rámec této publikace, nicméně pro představu je vhodné zmínit alespoň zhruba jeho základní postup. Hlavním úkolem kvantového obvodu je najít periodu $r$ funkce $f(x) = a^x \bmod N$, kde $a$ je vhodně zvolené celé číslo a $N$ je faktorizané číslo. Nalezená perioda $r$ musí být sudá. Pokud tomu tak není, je zapotřebí zvolit jinou hodnotu $a$ a opět najít periodu funkce $f(x)$. Jakmile je nalezena sudá perioda $r$, pak lze faktory jednoduše vypočítat jako největší společný dělitel čísel $(a^{r/2} + 1)$ a $N$ a čísel $(a^{r/2} - 1)$ a $N$. Podrobnosti lze nalézt v [15].

Kvantový obvod je tedy zapotřebí pouze pro nalezení periody funkce $f(x)$. Přípravu vhodné hodnoty $a$ a vypočtení hledaných faktorů z periody $r$ pak může zajistit obsluhující klasický počítač.

Jednoduchý příklad kvantového obvodu pro nalezení periody funkce $f(x)$ pro faktorizaci čísla $N = 35$ je zobrazen na obr. 11. Jako vhodné číslo bylo zvoleno $a = 6$, takže funkce $f(x) = 6^x \bmod 35$.




Obrázek 11: Kvantový obvod pro nalezení periody funkce $f(x) = 6^x \bmod 35$.

```
# inicializace
import matplotlib.pyplot as plt
import numpy as np

# import Qiskitu
from qiskit import IBMQ, Aer
from qiskit.providers.ibmq import least_busy
```

```
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister, transpile, assemble
from qiskit.circuit.library import C3XGate

# import zakladnich vizualizacnich nastroju
from qiskit.visualization import plot_histogram

n = 2 # pocet qubitu, na ktere bude aplikovana funkce 6^x mod 35
m = 3 # pocet qubitu pro funkci 6^x mod 35

def podminene_nasobeni_6mod35(mocnina):
    """Podminene nasobeni (vstup x 6) mod 35"""
    ko_fx = QuantumCircuit(3)
    for iteration in range(mocnina):
        ko_fx.swap(0,2)
        ko_fx.x(1)
    ko_fx = ko_fx.to_gate()
    ko_fx.name = "6^%i\nmod\n35" % (mocnina)
    podmineny_kofx = ko_fx.control()
    return podmineny_kofx

def qft_inverzni(n):
    """Inverzni kvantova Fourierova transformace"""
    ko_qft = QuantumCircuit(n)
    for qubit in range(n//2):
        ko_qft.swap(qubit, n-qubit-1)
    for j in range(n):
        for m in range(j):
            ko_qft.cp(-np.pi/float(2**(j-m)), m, j)
        ko_qft.h(j)
    ko_qft.name = "inverzni\nQFT"
    return ko_qft

# Je zapotrebi vytvorit kvantovy obvod s n+m qubity
# Taky bude zapotrebi n klasickych bitu, do kterych se bude
# zapisovat namereny vystup
ko = QuantumCircuit(n+m, n)

# Nastaveni vsech q0 a q1 do stavu superpozice |+>
for i in range(n):
    ko.h(i)

# Nastaveni vstupu funkce 6^x mod 35 na |001>
ko.x(n)

# Aplikace funkce 6^x mod 35 na superponovane q0 a q1
for i in range(n):
    ko.append(podminene_nasobeni_6mod35(2**i),
              [i] + [j+n for j in range(3)])

ko.barrier()

# Inverzni kvantova Fourierova transformace
ko.append(qft_inverzni(n), range(n))

ko.barrier()

# Mereni kvantoveho stavu qubitu
for i in range(n):
    ko.measure(i, i)

ko.draw()  # Zobrazeni
           # obvodu
```

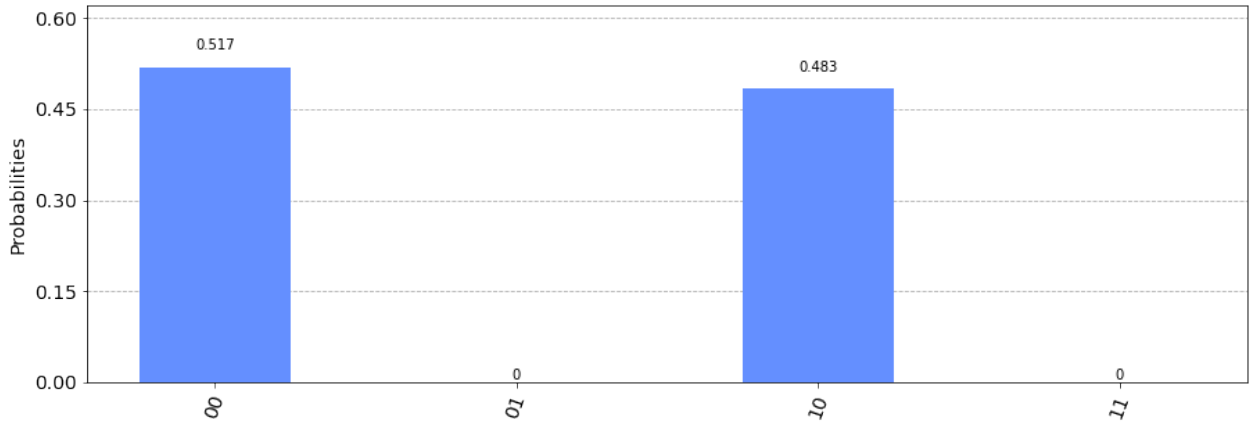
Zdrojový kód 6: Implementace kvantového obvodu pro nalezení periody funkce $f(x) = 6^x \bmod 35$ v Qiskitu.

Na simulátoru pak lze tento obvod spustit opět pomocí stejného zdrojového kódu 2. Výsledek je zobrazen v obr. 12.

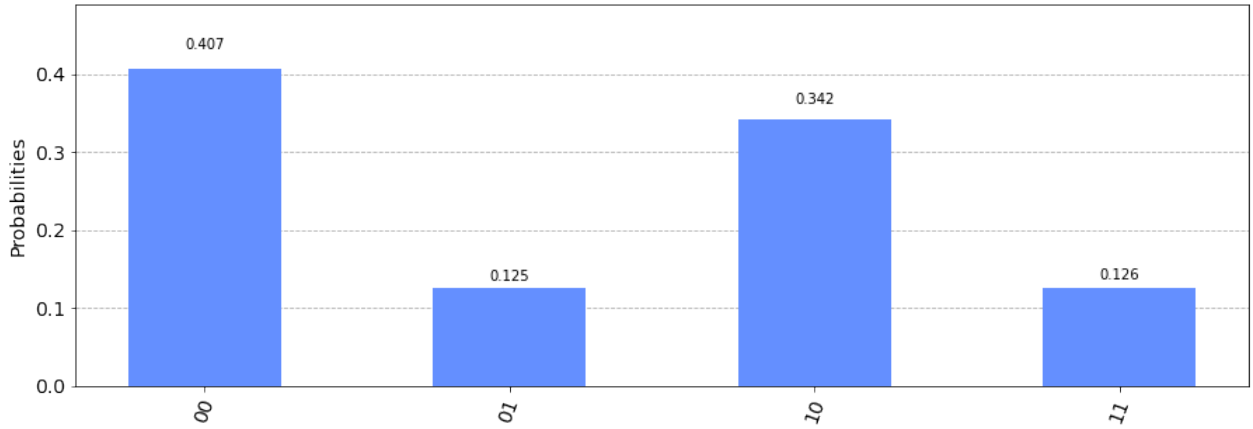
Na kvantovém počítači pak lze tento obvod spustit opět pomocí stejného zdrojového kódu 3. Výsledek je zobrazen v obr. 13.

Hledaná perioda se pak z výsledného rozložení pravděpodobností určí pomocí tzv. pokračujících zlomků, nicméně v tomto případě ji lze určit na první pohled jako počet vrcholů




Obrázek 12: Výsledné rozložení pravděpodobností naměřeného výstupu kvantového obvodu z obr. 11 po spuštění na IBM simulátoru (*ibmq_qasm_simulator*).




Obrázek 13: Výsledné rozložení pravděpodobností naměřeného výstupu kvantového obvodu z obr. 11 po spuštění na kvantovém počítači IBM (*ibmq_quito*).

tohoto rozložení, čili $r = 2$. Potom největší společný dělitel čísel $a^{r/2} + 1 = 6^{2/2} + 1 = 7$ a $N = 35$ je číslo 7 a čísel $a^{r/2} - 1 = 6^{2/2} - 1 = 5$ a $N = 35$ je číslo 5. Tím bylo faktorizováno číslo 35 na faktory 7 a 5.

Samozřejmě, toto je velmi jednoduchý příklad a proto vycházejí až triviální výsledky, nicméně například už při faktorizaci čísla 247 má funkce $f(x)$ při zvoleném $a = 8$ periodu $r = 12$ a potom $a^{r/2} \pm 1 = 8^{12/2} \pm 1$ vychází na čísla 262143 a 262145, jejichž největší společný dělitel s číslem 247 pak dává hledané faktory 13 a 19.

## Reference

[1] M. A. Nielsen and I. L. Chuang. *Quantum computation and quantum information*. Quantum computation and quantum information. 2000.

[2] Emma Strubell. An introduction to quantum algorithms. 2011.

[3] R. P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6-7):467–488, 1982.

[4] Peter W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, 1994.

[5] R. L. Rivest, A. Shamir, and L. M. Adleman. *A method for obtaining digital signatures and public key cryptosystems*, pages 217–239. Secure Communications and Asymmetric Cryptosystems. 2019.

[6] Andreas Baumhof. Breaking rsa encryption – an update on the state-of-the-art, 2019. Last access: April 19, 2022.

[7] Stephen Jordan. Quantum algorithm zoo, 2021. Last updated: February 1, 2021.

[8] MD SAJID ANIS et all. Qiskit: An open-source framework for quantum computing, 2021.

[9] John Polkinghorne; přeložil Pavel Cejnar. *Kvantová teorie - průvodce pro každého*. Praha: Dokořán. 2007.

[10] George Johnson; přeložili Jiří Podolský a Pavel Cejnar. *Zkratka napříč časem - cesta ke kvantovému počítači.* Praha: Argo. 2004.

[11] Pavel Cejnar. https://ipnp.cz/cejnar/publikace/qcomp.htm. Last access: April 19, 2022.

[12] Benjamin Schumacher. Quantum coding. *Phys. Rev. A*, 51:2738–2747, Apr 1995.

[13] A. Barenco, C. H. Bennett, R. Cleve, D. P. Divincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter. Elementary gates for quantum computation. *Physical Review A*, 52(5):3457–3467, 1995.

[14] Alexei Y. Kitaev, Alexander Shen, and Mikhail N. Vyalyi. Classical and quantum computation. In *Graduate studies in mathematics*, 2002.

[15] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.




This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 951732. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Germany, Bulgaria, Austria, Croatia, Cyprus, the Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, the United Kingdom, France, the Netherlands, Belgium, Luxembourg, Slovakia, Norway, Switzerland, Turkey, the Republic of North Macedonia, Iceland, and Montenegro. This project has received funding from the Ministry of Education, Youth and Sports of the Czech Republic (ID:MC2101).