دانشگاه صنعتی اصفهان
دانشکده فیزیک

گزارش پروژه

درس
مکانیک کوانتومی ۲

عنوان

مقدمه‌ای بر برنامه‌نویسی کوانتومی با کیت توسعه نرم‌افزار Qiskit

نگارنده

محمدحسین سلیمی

استاد درس

دکتر مهدی عبدی

بهار ۱۴۰۰

چکیده

زبان‌های برنامه‌نویسی کوانتومی ابزاری هستند که با استفاده از آن‌ها می‌توان ایده‌های مختلف را به سری دستوراتی تبدیل کرد که کامپیوترهای کوانتومی قادر به اجرای آن‌ها باشند. نه تنها آن‌ها برای کار با کامپیوترهای کوانتومی نیاز هستند، بلکه باعث کشف و توسعه الگوریتم‌های کوانتومی، حتی پیش از به وجود آمدن سخت‌افزار با قابلیت اجرای آن‌ها، نیز شده‌اند. از این زبان‌ها برای کنترل دستگاه‌های موجود، ارزیابی بازدهی الگوریتم‌های مختلف بر روی دستگاه‌های در دست تولید، کالیبرازیسیون دستگاه‌ها، آموزش مفاهیم محاسبات کوانتومی و ساخت انواع مختلف الگوریتم‌های کوانتومی استفاده می‌شود. در این گزارش قصد دارم تا با معرفی یکی از چارچوب‌های برنامه‌نویسی کوانتومی، Qiskit ، با مفاهیم کلی برنامه‌نویسی کوانتومی آشنا شویم، الگوریتم کوانتومی‌ای را پیاده سازی و کد آن را اجرا کنیم.

فهرست مطالب

۱.۰ مقدمه

در سال ۲۰۱۷ شرکت IBM برای اولین بار یک کیت توسعه نرم‌افزارِ(SDK) متن باز(open-source) برای محاسبات کوانتومی به نام Qiskit را معرفی کرد. Qiskit ابزارهایی برای خلق و دست‌کاری برنامه‌های کوانتومی در اختیار کاربر قرار می‌دهد و این اجازه را می‌دهد تا کاربر بر روی یک کامپیوتر کوانتومی شبیه‌سازی شده بر روی سیستم خودش، کارایی کدش را بررسی کند.

نسخه اصلی این کیت از زبان برنامه‌نویسی پایتون استفاده می‌کند که کار را برای کاربران کمی ساده‌تر می‌کند. چرا که در حالت کلی برای اجرای دستوراتی بر روی کامپیوترها باید از زبان‌های سطح پایین، مانند Assembly ، استفاده کرد. کامپیوترهای کوانتومی نیز از این قاعده مستثنی نیستند. Qiskit این قابلیت را فراهم می‌آورد تا با استفاده از یک زبان سطح بالا، پایتون، کاربر دستورات خود را به ماشن بفهماند.(منظور از زبان سطح بالا زبانی است که از زبان ماشین دورتر است و کاربران راحت‌تر با آن‌ها کار می‌کنند.) البته نسخه میکرویی از این کیت وجود دارد که از زبان‌های جاوااسکریپت و سوئیفت نیز پشتیبانی می‌کند.

یکی از بزرگترین مزیت‌های استفاده از این کیت، داشتن دسترسی به کامپیوترهای کوانتومی شرکت IBM است. این شرکت یک پلتفرم آنلاین به اسم تجربه‌ی کوانتومی آی‌بی‌ام (IBM-Q-Experience) به وجود آورده است که به کاربران این اجازه را می‌دهد تا برنامه‌های کوانتومی نوشته شده خود با Qiskit را بر روی کامپیوترهای کوانتومی واقعی اجرا کنند. در نسخه رایگان این پلتفرم، کاربران به ۱۴ کامپیوتر کوانتومی که حداکثر تا ۱۵ کیوبیت دارند،دسترسی دارند.

۲.۰ گیت کوانتومی

۱.۲.۰ گیت‌های پاولی

در پایین‌ترین سطح، الگوریتم‌های کوانتومی از یک سری پایه‌ها و یا بنیادها ساخته شده‌اند. درست مانند اگوریتم‌های کلاسیکی که در پایین‌ترین سطح از گیت‌های AND ، NOT و OR ساخته شده‌اند. در این بخش قصد داریم تا کمی با این پایه‌ها آشنا شویم.

یک سیستم کوانتومی ایده‌آل از n کیوبیت که با حالت مختلط C^{2^n} تعریف می‌شود، تشکیل می‌شود. برای مثال، حالت یک سیستم تک کیوبیتی را می‌توان به صورت

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \qquad (۱)$$

نوشت. که در آن α و β اعدادی مختلط و $|0\rangle = \binom{1}{0}$ و $|1\rangle = \binom{0}{1}$ هستند.

انجام محاسبات در الگوریتم‌های کوانتومی در اصل انجام یک سری تحولات ریاضی بر روی بردار حالت است. این تحولات به واسطه ماتریس‌های یکانی مختلط $2^n \times 2^n$ صورت میگیرد که به آن‌ها گیت‌های کوانتومی گفته می‌شود.

حالت یک کیوبیت از $|0\rangle$ به $|1\rangle$ زمانی تغییر میکند که گیت کوانتومی یا عملگر X بر روی آن اثر کند که به صورت زیر تعریف می‌شود:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}. \tag{۲}$$

تحولات اعمالی بر روی یک کیوبیت از یک سیستم چند کیوبیتی را می‌توان با ضرب تانسوری X و ماتریس همانی، $\mathbb{1}$، بدست آورد. به طور مثال، تاثیر اعمال X بر روی دومین کیوبیت از یک سیستم سه کیوبیتی به صورت $\mathbb{1} \otimes X \otimes \mathbb{1}$ خواهد بود. عملگر X یکی از سه عملگر پاولی است که به همراه ماتریس همانی، پایه‌ی تمام تحولات یکانی بر روی تک کیوبیت را تشکیل می‌دهند. نمایش ماتریسی دو عملگر دیگر Y و Z به صورت زیر است:

$$Y = \begin{pmatrix} 0 & -i \\ -i & 0 \end{pmatrix},\ Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{۳}$$

۲.۲.۰ گیت آدامار

این عملگر که بر روی یک کیوبیت عمل میکند، حالت پایه $|0\rangle$ را به حالت $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ و حالت پایه $|1\rangle$ را به حالت $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$ می‌برد. در نتیجه اعمال این عملگر روی یک حالت پایه، برهم‌نهی کوانتومی به وجود می‌آورد که در صورت اندازه‌گیری، با احتمالی برابر، می‌تواند $|0\rangle$ و یا $|1\rangle$ شود. نمایش ماتریسی آن به صورت زیر است:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}. \tag{۴}$$

۳.۲.۰ گیت CX

این گیت بر روی دو کیوبیت اثر می‌کند. بدین صورت که یکی از کیوبیت‌ها را به عنوان کنترل و دیگری را به عنوان هدف در نظر میگیرد. در صورتی که کیوبیت کنترل در حالت $|1\rangle$ باشد،این گیت عملگر X را بر روی کیوبیت هدف اثر می‌دهد. اگر کیوبیت کنترل در حالت برهم‌نهی (superposition) باشد، آنگاه این گیت باعث به وجود آمدن درهم تنیدگی بین دو کیوبیت می‌شود. در بعضی مراجع به این گیت CNOT و یا controlled-NOT نیز می‌گویند.

نمایش ماتریسی آن به صورت زیر است:

$$CX = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \quad (۵)$$

۳.۰ سلام دنیا!!!

یکی از سنت‌های یادگیری زبانی جدید در بین برنامه‌نویسان، نوشتن برنامه‌ای به نام hello-world است. بدین صورت که برنامه‌ای خیلی ساده و گاها تک خطی نوشته می‌شود که فقط یک جمله، hello-world ، را چاپ می‌کند. هدف از انجام این کار، بررسی نصب صحیح اجزای زبان بر روی سیستم و یادگیری اجرای برنامه‌ها در زبان جدید است. در این بخش قصد داریم تا با پیروی از این سنت، یک hello-world کوانتومی با استفاده از Qiskit بنویسیم و آن را اجرا کنیم.

در این برنامه کوچک یک مدار کوانتومی دو کیبویتی تشکیل می‌دهیم، درهم تنیدگی به وجود می‌آوریم و احتمال را محاسبه می‌کنیم.

۱.۳.۰ نصب

پیش از هر کاری باید Qiskit را نصب کنیم. همانطور که گفته شد، این کیت بر روی پایتون سوار است و از امکانات این زبان استفاده می‌کند، پس باید پایتون بر روی سیستم وجود داشته باشد. اگر از سیستم‌های یونیکسی(گنو/لینوکس، مک) استفاده می‌کنید، پایتون به صورت پیش فرض روی سیستم‌تان وجود دارد. اگر کاربر ویندوز هستید، می‌توانید به مراجعه به سایت رسمی پایتون به نشانی www.python.org، اقدام به نصب آن کنید.

این کیت به صورتی کتابخانه‌ای قابل نصب بر روی پکیج منیجر پایتون، pip، وجود دارد. پیشنهاد می‌شود که قبل از نصب Qiskit ، ابتدا یک محیط مجازی پایتون(virtualenv) بر روی سیستم خود به وجود آورید و پس از فعال کردن آن اقدام به نصب Qiskit کنید. برای نصب Qiskit از دستور ساده زیر در محیط ترمینال می‌توانید استفاده کنید.

```
pip install qiskit
pip install matplotlib
```

پکیج matplotlib برای نمایش نمودارها لازم است.

۲.۳.۰ ساخت مدار

با فراخوانی پکیج‌های مورد نیاز شروع می‌کنیم

```
from qiskit import *
import matplotlib
from qiskit.visualization import plot_histogram
from qiskit.tools.monitor import job_monitor
```

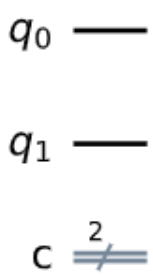
سپس باید یک مدار کوانتومی با دو کیوبیت و دو بیت کلاسیکی تشکیل دهیم. بیت‌های کلاسیکی برای ذخیره نتایج مشاهده شده استفاده می‌شوند.

```
circuit = QuantumCircuit(2,2)
```

با استفاده از دستور

```
circuit.draw(output='mpl')
```

می‌توان تصویر مدار ساخته شده را دید که به صورت زیر خواهد بود.



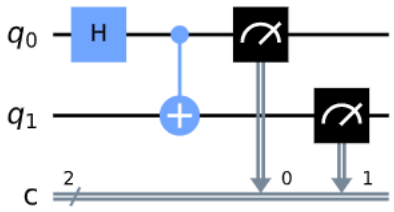
شکل ۱ – مدار کوانتومی شامل دو کیوبیت و دو بیت کلاسیکی

اکنون به مدار ساخته شده گیت‌ها را اضافه می‌کنیم. دو کیوبیت تشکیل شده در مدار داری اندیس‌های صفر و یک هستند که به ترتین کیوبیت اول و دوم را مشخص می‌کنند.

گیت آدامار را بر روی کیوبیت با اندیس صفر اضافه می‌کنیم. سپس گیت CX را اعمال میکنیم. ترتیب به این صورت است که عدد اول، اندیس کیوبیت کنترل و عدد دوم اندیس کیوبیت هدف است. سپس اندازه‌گیری بر روی کیوبیت‌های صفر و یک انجام می‌شود و نتایج در بیت‌های کلاسیکی صفر و یک دخیره می‌شود.

```
circuit.h(0)
circuit.cx(0,1)
circuit.measure([0,1], [0,1])
circuit.draw(output='mpl')
```

و در آخر دوباره تصویر مدار ساخته می‌شود.



شکل ۲ – مدار کوانتومی نهایی

۳.۳.۰ اجرا بر روی شبیه‌ساز

از مزیت‌های خوب Qiskit ، همراه شدن آن با شبیه‌سازهای موضعی (local) است. به این منظور که کاربران بر روی سیستم خود می‌توانند یک کامپیوتر کوانتومی تقریبا ایده‌آل داشته باشند. در اینجا قصد داریم تا با استفاده از qasm-simulator که یک موتور شبیه‌سازی است که همراه Qiskit بر روی سیسنم کاربر نصب می‌شود، کد نوشته شده خود را اجرا کنیم.

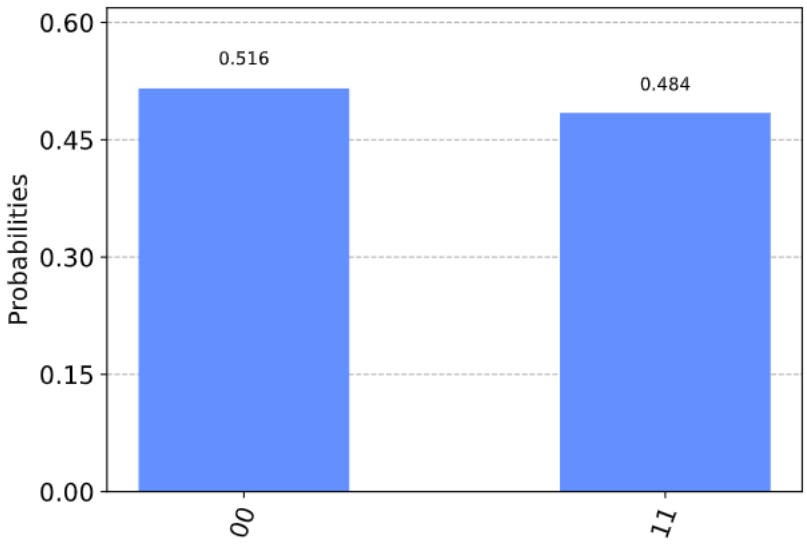
ابتدا موتور شبیه‌ساز را تعریف می‌کنیم

```
simulator = Aer.get_backend('qasm_simulator')
```

سپس مدار را بر روی این شبیه‌ساز اجرا می‌کنیم و نتایج به دست آمده را در متغیر$result$ ذخیره می‌کنیم. در آخر نمودار نتایج به دست آمده از شبیه‌سازی را رسم می‌کنیم.

```
result = execute(circuit, backend=simulator).result()
plot_histogram(result.get_counts(circuit))
```

در آخر نمودار نتایج به دست آمده از شبیه‌سازی را رسم می‌کنیم.



شکل ۳ – نمودار احتمال

۴.۳.۰ اجرا بر روی کامپیوتر کوانتومی واقعی

اکنون می‌خواهیم برنامه خود را بر روی یک کامپیوتر کوانتومی واقعی اجرا کنیم و نتیجه آن را با نتیجه قسمت قبل مقایسه کنیم.

برای این کار ابتدا با مراجعه به سایت quantum-computing.ibm.com یک حساب کاربری ایجاد می‌کنیم. با مراجعه به قسمت پروفایل حساب کاربری، API-token را کپی کرده و با استفاده از دستور پایتونی زیر سیستم خود را به حساب کاربری IBM متصل می‌کنیم.

```
IBMQ.save_account('<API token>')
```

اکنون می‌توانیم حساب کاربری خود را درون کد فراخوانی کنیم و با استفاده از آن به کامپیوترها کوانتومی شرکت IBM متصل شویم. سپس نوع کامپیوتری که میخواهیم به آن متصل شویم را مشخص می‌کنیم و در مرحله بعد نام کامپیوتر را تعریف می‌کنیم. در اینجا ibmq-۱۶-melbourne نام کامپیتر مورد استفاده است.

```
IBMQ.load_account()
provider = IBMQ.get_provider(hub = 'ibm-q')
qcomp = provider.get_backend('ibmq_16_melbourne')
```

اکنون مدار را بر روی کامپیوتر مورد نظر اجرا می‌کنیم.

```
job = execute(circuit, backend=qcomp)
```

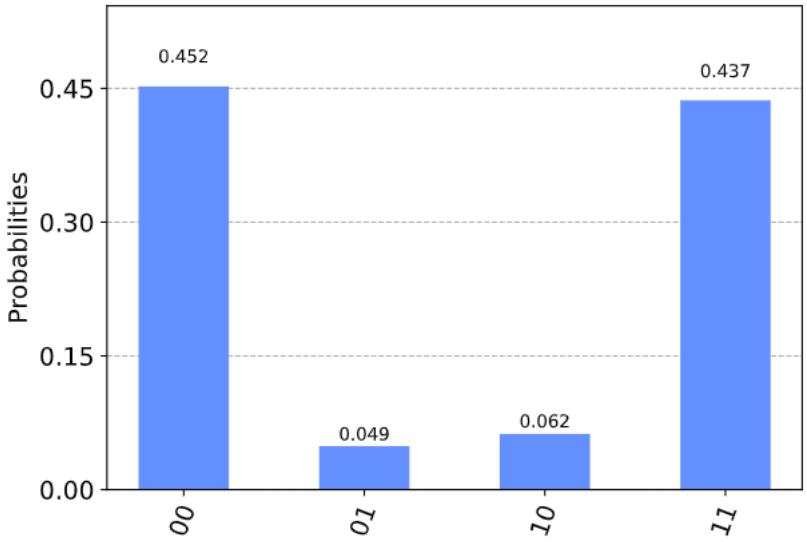
با توجه به این که این کامپیوترها در اختیار عموم قرار دارند، ممکن است مدتی در صف منتظر بمانیم تا کد اجرا شود. با دستور زیر می‌توانیم از وضعیت job مطلع شویم. این دستور به صورت اتوانیک خود را آپدیت میکند.

```
job_monitor(job)
```

پس از انجام job ، نتیجه به دست آمده را در متغیر $result$ ذخیره می‌کنیم و نمودار احتمال را رسم می‌کنیم.

```
result = job.result()
plot_histogram(result.get_counts(circuit))
```

نمودار نهایی به صورت زیر است که همانگونه که مشاهده میکنید با قسمت قبل متفاوت است. دلیل این تفاوت نیز این است که کامپیوترهای کوانتومی واقعی درصدی خطا دارند که ممکن است در اثر نویزهای سیستمی، کالیبره نبودن دستگاه و یا دلایلی دیگر باشد.



شکل ۴ – نمودار احتمال بدست آمده از کامپیوتر کوانتومی واقعی

۵.۳.۰ یک تکه کد

از آنجایی که ممکن است صف‌های طولانی‌ای برای استفاده از دستگاه‌ها شکل بگیرد، می‌توان با اجرا کردن تکه کد زیر خلوت‌ترین کامپیوتر کوانتومی را پیدا کرد و از آن استفاده کرد.

با تغییر مقدار متغییر num_qubits می‌توان حد پایین تعداد کیوبیت‌های دستگاه را مشخص کرد.

```
num_qubits = 2

from qiskit.providers.ibmq import least_busy
possible_devices = provider.backends(filters=lambda x:
                                     x.configuration().n_qubits >= num_qubits
                                       and
                                     x.configuration().simulator == False)
qcomp = least_busy(possible_devices)
print(qcomp)
```

۴.۰ مثالی دیگر

به عنوان مثالی دیگر، در این بخش به معرفی الگوریتم برنستین–وزیرانی می‌پردازیم و آن را در Qiskit پیاده سازی می‌کنیم.

۱.۴.۰ الگوریتم برنستین–وزیرانی

فرض کنیم یک جعبه سیاه حاوی یک رشته کاراکتر داریم که میخواهیم آن را حدس بزنیم. کاراکترهای این رشته فقط می‌توانند ۱ و یا ۰ باشند. خاصیت این جعبه آن است که رشته کاراکتر دریافتی از ما را با رشته کاراکتر درون

خودش بیت به بیت مقایسه می‌کند. یعنی اگر رشته دارای سه کاراکتر باشد، به ترتیب از راست به چپ کاراکترها را تک به تک باهم مقایسه میکند و اگر دو کاراکتر ۱ بودند، ۱ برمیگرداند و در غیر این صورت ۰ بازگردانده می‌شود.

یک کامپیوتر کلاسیکی، پس از n بار حدس، می‌تواند یک رشته کاراکتر n تایی را حدس بزند. به عنوان مثال، فرض کنید که رشته درون جعبه 110 است. حدس اول کامپیوتر کلاسیکی اگر 001 باشد، مشخص میشود که کاراکتر اول از سمت راست رشته درون جعبه صفر است. اگر حدس بعدی آن 010 باشد، مشخص می‌شود که کاراکتر دوم، یک است. به همین ترتیب آخرین کاراکتر نیز معلوم می‌شود و رشته حدس زده می‌شود.

همین مسئله را یک کامپیوتر کوانتومی با استفاده از الگوریتم برنستین–وزیرانی میتواند در یک حدس حل کند. در حالت کلی، فرم ریاضی این الگوریتم به صورت زیر است:

$$|00\dots0\rangle \xrightarrow{H^{\otimes n}} \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle \xrightarrow{f_s} \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} (-1)^{s\cdot x}|x\rangle \xrightarrow{H^{\otimes n}} |s\rangle. \qquad (۶)$$

که در آن f_s جعبه حاوی رشته کاراکتر s است. به عنوان مثال فرض کنید $n=2$ کیوبیت داریم و رشته درون جعبه $s=11$ است. کیوبیت‌ها در حالت اولیه $|\psi\rangle_0 = |00\rangle$ قرار دارند. با اعمال گیت آدامار بر روی دو کیوبیت خواهیم داشت:

$$|\psi_1\rangle = \frac{1}{2}\left(|00\rangle + |01\rangle + |10\rangle + |11\rangle\right) \qquad (۷)$$

اکنون باید تاثیر جعبه بر روی کیوبیت‌ها را اعمال کنیم. که برای $s=11$ به صورت زیر خواهد بود:

$$|x\rangle \xrightarrow{f_s} (-1)^{x\cdot 11}|x\rangle. \qquad (۸)$$

پس

$$|\psi_2\rangle = \frac{1}{2}\left((-1)^{00\cdot 11}|00\rangle + (-1)^{01\cdot 11}|01\rangle + (-1)^{10\cdot 11}|10\rangle + (-1)^{11\cdot 11}|11\rangle\right) \qquad (۹)$$

که پس از ساده سازی خواهیم داشت:

$$|\psi_2\rangle = \frac{1}{2}\left(|00\rangle - |01\rangle - |10\rangle + |11\rangle\right) \qquad (۱۰)$$

با اعمال دوباره گیت آدامار خواهیم داشت:

$$|\psi_3\rangle = |11\rangle \qquad (۱۱)$$

که با اندازه گیری به $s = 11$ خواهیم رسید.

۲.۴.۰ پیاده سازی در Qiskit

مانند برنامه‌ی سلام دنیا، با فراخوانی پکیج‌های مورد نیاز آغاز می‌کنیم

```
from qiskit import *
import matplotlib
from qiskit.visualization import plot_histogram
from qiskit.tools.monitor import job_monitor
```

رشته s را تعریف می‌کنیم و بر اساس آن مدار را می‌سازیم

```
s = '1101'
n = len(s)
circuit = QuantumCircuit(n+1,n)
```

تعداد کیوبیت‌ها را یکی بیشتر از تعداد کاراکترهای رشته در نظر می‌گیریم. در Qiskit کیوبیت‌ها دارای حالت اولیه $|0\rangle$ هستند، پس فقط بر روی کیوبیت آخر عملگر X را اعمال می‌کنیم. از کیوبیت آخر برای به وجود آوردن جعبه استفاده می‌کنیم. بر روی تمام کیوبیت‌ها گیت آدامار را اثر میدهیم. سپس جعبه را می سازیم. بدین صورت که عملگر CX را بر روی کیوبیت‌هایی که شامل کاراکتر ۱ هستند و کیوبیت آخر اثر می‌دهیم. این که را در جهت معکوس انجام می‌دهیم. بدین صورت که اگر اولین کاراکتر رشته ۱ بود، یک عملگر CX بر روی n امین کیوبیت و کیوبیت آخر اعمال می‌کنیم. سپس دوباره بر روی تمامی کیوبیت‌ها گیت آدامار را اثر می‌دهیم و احتمال را اندازه‌گیری می‌کنیم

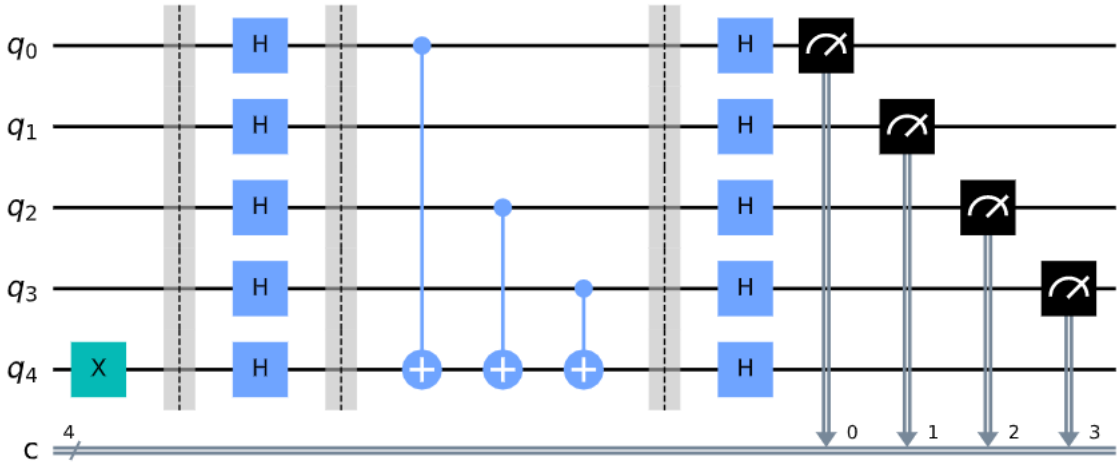
```
circuit.x(n)
circuit.barrier()
circuit.h(range(n+1))
circuit.barrier()
for ii, yesno in enumerate(reversed(s)):
    if yesno == '1':
        circuit.cx(ii,n)

circuit.barrier()
circuit.h(range(n+1))
circuit.measure(range(n), range(n))
```

با استفاده از دستور

```
circuit.draw(output='mpl')
```

مدار را جهت مشاهده رسم می‌کنیم که به صورت زیر در می‌آید. خطوط عمودی‌ای که بر روی مدار وجود دارند صرفا به منظور تفکیک قسمت‌های مختلف مدار به هنگام مشاهده شکل مدار است.

همانند قسمت قبل، برای اجرای مدار یک بار از شبیه ساز موضعی و یک بار از کامپیوتر کوانتومی واقعی استفاده خواهیم کرد.



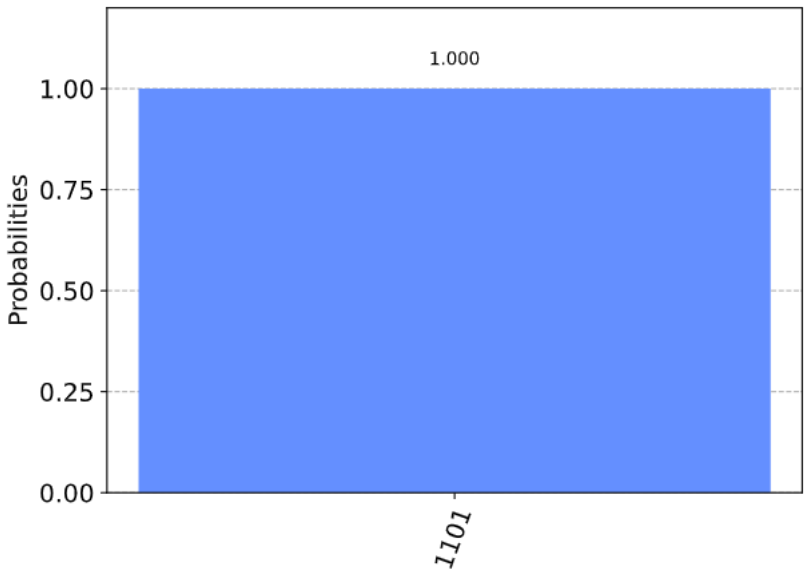
شکل ۵ – مدار کوانتومی الگوریتم برنستین–وزیرانی

۳.۴.۰ اجرا بر روی شبیه‌ساز

تعداد دفعات انجم آزمایش را برابر یک می‌گذاریم تا بیشتر انجام نشود.

```
simulator = Aer.get_backend('qasm_simulator')
result = execute(circuit, backend=simulator, shots=1).result()
plot_histogram(result.get_counts(circuit))
```

نتیجه به دست آمده به صورت زیر است. در یک بار آزمایش، نتیجه به دست آمده 1101 است که برابر است با رشته‌ای که از اول تعریف کرده بودیم.



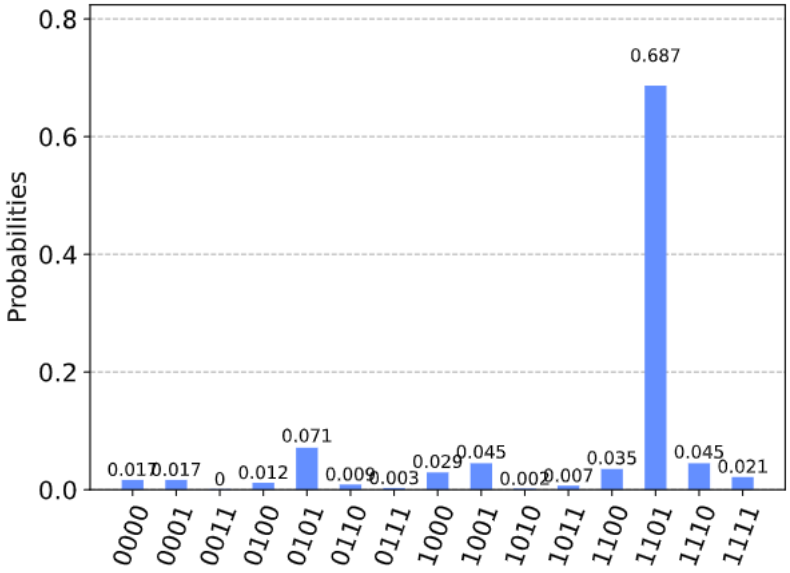
شکل ۶ – نتیجه شبیه سازی موضعی

۴.۴.۰ اجرا بر روی کامپیوتر کوانتومی واقعی

مانند قسمت قبل کد را بر روی کامپیوتر کوانتومی واقعی اجرا می‌کنیم

```
IBMQ.load_account()
provider = IBMQ.get_provider(hub='ibm-q')
qcomp = provider.get_backend('ibmq_16_melbourne')
job = execute(circuit,backend=qcomp)
job_monitor(job)
resualt = job.result()
plot_histogram(resualt.get_counts(circuit))
```

که نتیجه آن به صورت زیر است. همانطور که مشخص است با یک بار انجام آزمایش، با احتمال نزدیک به ۷۰



شکل ۷ – نتیجه بدست آمده از کامپیوتر کوانتومی واقعی

درصد رشته کاراکتر درست را بدست آورده است.

۵.۰ سخن پایانی

چارچوب‌ها و کیت‌های توسعه نرم‌افزار و حتی زبان‌های گوناگونی برای برنامه‌نویسی کوانتومی وجود دارند و Qiskit فقط یکی از آن‌هاست. هر کدام از این تکنولوژی‌ها قابلیت‌ها، مزایا و معایب خود را دارند. اما همه‌شان یک وجه مشترک دارند، و آن هم ارزش حداقل یک‌بار استفاده از آن‌هاست. استفاده از این تکنولوژی‌ها به درک بسیاری از مطالب کوانتومی و علوم کامپیوتر کمک می‌کند و باعث می‌شود بیشتر به دنبال یادگیری باشیم. کامپیوترهای کوانتومی روز به روز پیشرفته‌تر می‌شوند و کار با آن‌ها راحت‌تر می‌شود. گسترش و یادگیری این تکنولوژی‌ها به پیشرفت این نوع کامپیوترها نیز کمک بسیاری می‌کنند.

تمامی کدهای توضیح داده شده در این گزارش به همراه jupyter-notebook های آن‌ها در یک مخزن گیت‌هاب به آدرس github.com/Perun21/q2-qiskit قابل دسترسی می‌باشند و می‌توانید از آن‌ها استفاده نمایید.

مراجع

[1] E. Bernstein, U. Vazirani, and S. J. Comput, "Quantum complexity theory *," *Society for Industrial and Applied Mathematics*, vol.26, p.7, 1997.

[2] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, H. Weinfurter, and I. Background, "Elementary gates for quantum computation november 1995," *PHYSICAL REVIEW A*, vol.52, 1995.

[3] B. Heim, M. Soeken, S. Marshall, C. Granade, M. Roetteler, A. Geller, M. Troyer, and K. Svore, "Quantum programming languages," *Nature Reviews Physics*, vol.2, pp.709–722, 12 2020.

[4] "Qiskit official documentation,"