Matematyka w informatyce przyszłości

Zaimplementowane programy wraz z wnioskami

Autor: Maciej Sandacz
Data oddania: 25 kwietnia 2013 r.

Podstawowe pojecia komputerow kwantowych - qbit, bramka kwantowa. Wstep do biblioteki libquantum

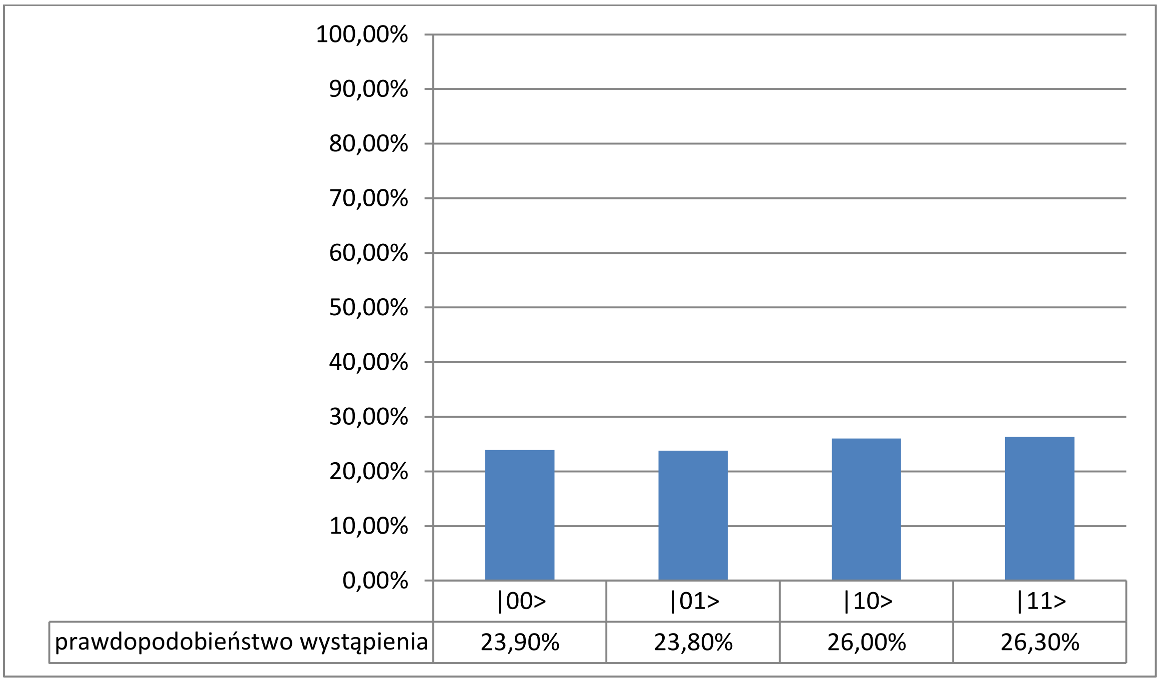
Podstawowe pomiary (ćw. 1 i 2)

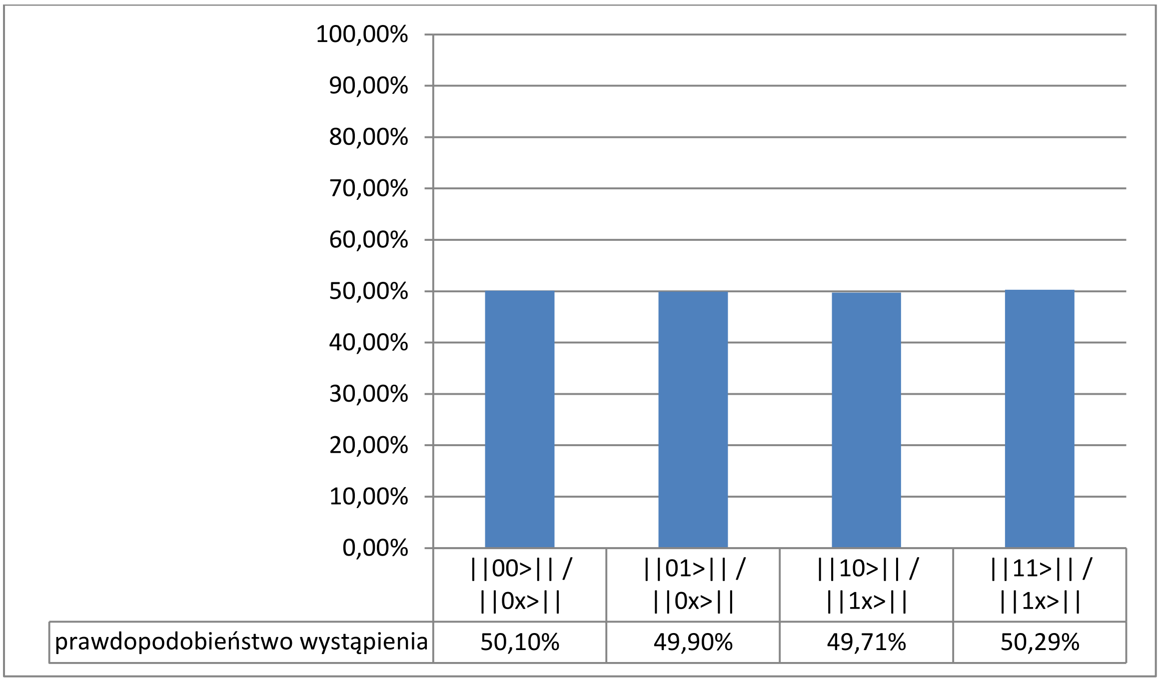
Kod (lab2-1.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <quantum.h>

int main ()
{
  int i;
  quantum_reg reg;
  MAX_UNSIGNED result;

  srand(time(0));

  reg = quantum_new_qureg(0, 2);

  quantum_hadamard(0, &reg);

  quantum_hadamard(1, &reg);


  for (i = 0; i < 1000; ++i) {

    result = quantum_measure(reg);

    printf("%i ", result);

  }

 return 0;
}
```

Wyniki

Program dokonuje 1000 pomiarów dwóch qubitów, na których zastosowano bramkę Hadamarda. Na podstawie otrzymanych wyników obliczone zostały odpowiednie warunkowe prawdopodobieństwa oraz histogram poszczególnych wystąpień.





Z otrzymanych wyników można wywnioskować, że wartości otrzymywane na poszczególnych qubitach są od siebie niezależne.

Bramka SWAP z bramek CNOT (ćw. 3)

Kod (lab2-2.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <quantum.h>

int main ()
{
```

```
  int i;
  quantum_reg reg;
  MAX_UNSIGNED result;

  srand(time(0));

  for (i = 0; i < 4; ++i) {

    reg = quantum_new_qureg(i, 2);

    quantum_cnot(1, 0, &reg);

    quantum_cnot(0, 1, &reg);

    quantum_cnot(1, 0, &reg);

    result = quantum_measure(reg);

    printf("%i -> %i \n", i, result);

  }

 return 0;
}
```

Wyniki

Bramka SWAP skonstruowana z trzech bramek CNOT daje oczekiwane rezultaty dla wszystkich wartości:

```
0 -> 0
1 -> 2
2 -> 1
3 -> 3
```

Bramka Toffoliego (ćw. 4)

Kod (lab2-3.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <quantum.h>

int main ()
{
  int i;
  quantum_reg reg;
  MAX_UNSIGNED result;

  srand(time(0));

  for (i = 0; i < 8; ++i) {

    reg = quantum_new_qureg(i, 3);

    quantum_toffoli(2, 1, 0, &reg);

    result = quantum_measure(reg);

    printf("%i -> %i \n", i, result);
```

```
  }

 return 0;
}
```

Wyniki

Wynik działania bramki jest zgodny z oczekiwaniami dla wszystkich wartości:

```
0 -> 0
1 -> 1
2 -> 2
3 -> 3
4 -> 4
5 -> 5
6 -> 7
7 -> 6
```

Obliczenia kwantowe

Stany splątane (ćw. 1)

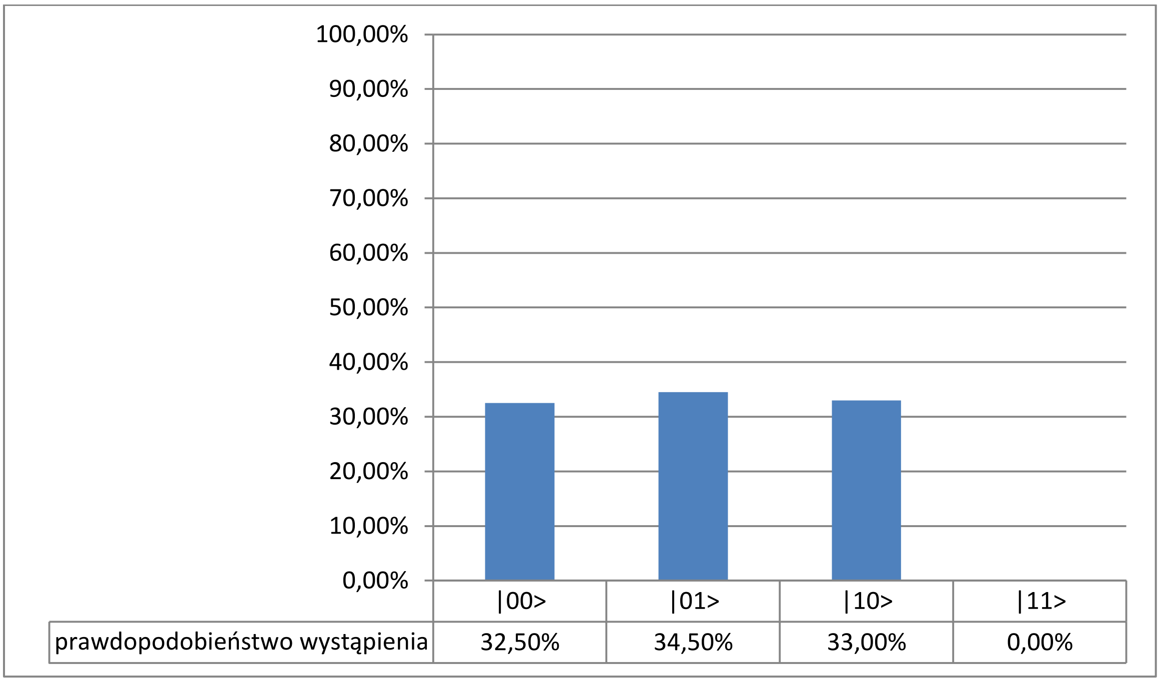
Kod (lab3-1.c)

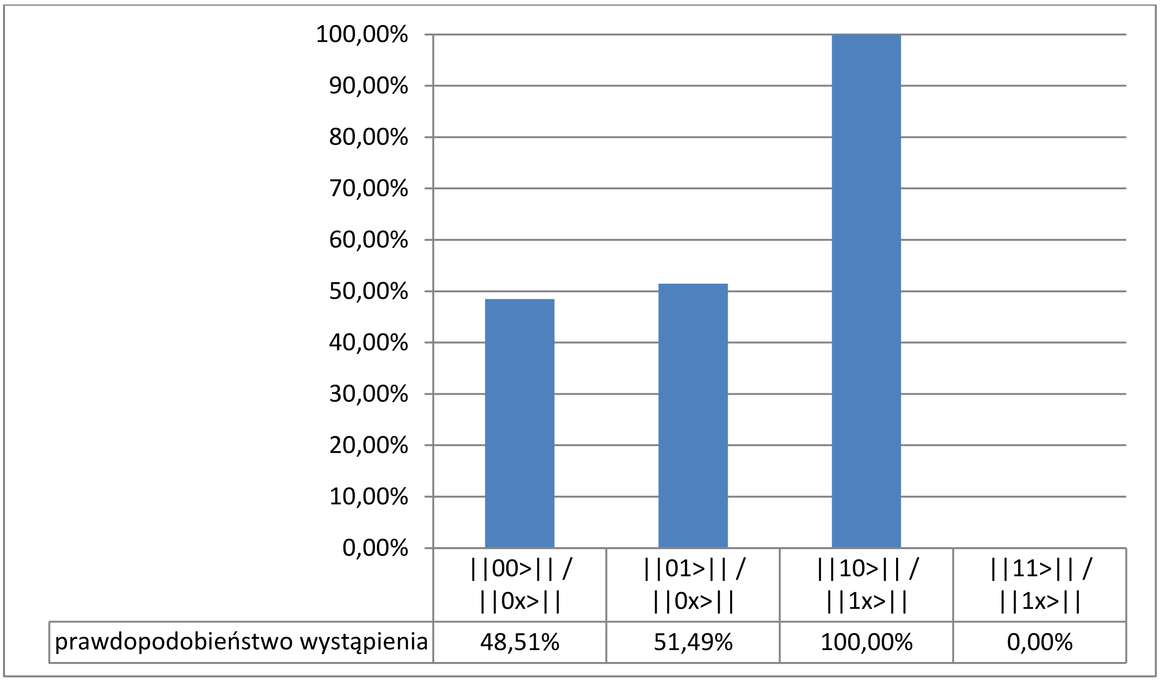
```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <quantum.h>
#include <math.h>

#define PI 3.14159265

void quantum_w(int target, quantum_reg *reg)
{
  quantum_matrix m = quantum_new_matrix(2, 2);

  m.t[0] =  sqrt(2. / 3);  m.t[1] = -sqrt(1. / 3);
  m.t[2] =  sqrt(1. / 3);  m.t[3] =  sqrt(2. / 3);

  quantum_gate1(target, m, reg);

  quantum_delete_matrix(&m);
}

void quantum_u(int target, quantum_reg *reg)
{
  quantum_matrix m = quantum_new_matrix(2, 2);

  m.t[0] =  cos(PI / 8);  m.t[1] =  sin(PI / 8);
  m.t[2] = -sin(PI / 8);  m.t[3] =  cos(PI / 8);

  quantum_gate1(target, m, reg);

  quantum_delete_matrix(&m);
}

void quantum_u_conjt(int target, quantum_reg *reg)
```

```
{
  quantum_matrix m = quantum_new_matrix(2, 2);

  m.t[0] =  cos(PI / 8);  m.t[1] = -sin(PI / 8);
  m.t[2] =  sin(PI / 8);  m.t[3] =  cos(PI / 8);

  quantum_gate1(target, m, reg);

  quantum_delete_matrix(&m);
}

void quantum_chadamard(int control, int target, quantum_reg *reg)
{
  quantum_u_conjt(target, reg);
  quantum_cnot(control, target, reg);
  quantum_u(target, reg);
}

quantum_reg quantum_new_spooky()
{
  quantum_reg reg = quantum_new_qureg(0, 2);

  quantum_w(1, &reg);
  quantum_chadamard(1, 0, &reg);
  quantum_hadamard(0, &reg);

  return reg;
}

int main ()
{
  int i;
  quantum_reg reg;
  MAX_UNSIGNED result;

  srand(time(0));

  reg = quantum_new_spooky();

  for (i = 0; i < 1000; ++i) {

    result = quantum_measure(reg);

    printf("%i ", result);

  }

  quantum_delete_qureg(&reg);

  return 0;
}
```

Wyniki

Skonstruowany stan splątany został poddany takim samym pomiarom, jak stan zbudowany w pierwszym zadaniu, a otrzymane wyniki zostały poddane takiej samej obróbce i przedstawione poniżej.





W odróżnieniu od wyników w pierwszym zadaniu, wyniki pomiarów poszczególnych qubitów okazały się od siebie zależne. Wartość |0> na lewym qubicie oznacza równomierny rozkład wartości na prawym qubicie, natomiast wartość |1> na lewym qubicie gwarantuje wartość |0> na prawym qubicie.

Problem Deutscha (ćw. 3)

Kod (lab3-2.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <quantum.h>
```

```c
COMPLEX_FLOAT a0 = 0.0f;
COMPLEX_FLOAT a1 = 1.0f;
int I = 1;
int O = 0;

void quantum_not(int target, quantum_reg *reg)
{
  quantum_matrix m = quantum_new_matrix(2, 2);

  m.t[0] = a0;  m.t[1] = a1;
  m.t[2] = a1;  m.t[3] = a0;

  quantum_gate1(target, m, reg);

  quantum_delete_matrix(&m);
}

void quantum_uf (int f, int input, int output, quantum_reg *reg)
{
  quantum_matrix m = quantum_new_matrix(2, 2);

  switch(f)
  {
    case 0:
    break;

    case 1:
      quantum_cnot(input, output, reg);
    break;

    case 2:
      quantum_not(output, reg);
      quantum_cnot(input, output, reg);
    break;

    case 3:
      quantum_not(output, reg);
    break;
  }

  quantum_delete_matrix(&m);
}

int main ()
{
  int i, f;
  quantum_reg reg;

  srand(time(0));

  printf("=============== TEST DZIAŁANIA FUNKCJI Ufi ===============\n\n");

  for (f = 0; f < 4; ++f) {

    for (i = 0; i < 4; ++i) {

      reg = quantum_new_qureg(i, 2);

      printf("Uf%i", f);
      quantum_print_qureg(reg);
```

```
      quantum_uf(f, I, O, &reg);

      printf(" =>");
      quantum_print_qureg(reg);

      quantum_delete_qureg(&reg);

    }

    printf("\n");

  }

  printf("====== TEST DZIAŁANIA ROZWIĄZANIA PROBLEMU DEUTSCHA ======\n\n");

  for (f = 0; f < 4; ++f) {

    reg = quantum_new_qureg(0, 2);

    quantum_not(0, &reg);
    quantum_not(1, &reg);

    quantum_hadamard(0, &reg);
    quantum_hadamard(1, &reg);

    quantum_uf(f, I, O, &reg);

    quantum_hadamard(1, &reg);

    printf("Uf%i\n", f);
    quantum_print_qureg(reg);

    quantum_delete_qureg(&reg);

  }

  return 0;
}
```

Wyniki

```
=============== TEST DZIAŁANIA FUNKCJI Ufi ===============
Uf0 1.000000 +0.000000i|0> (1.000000e+00) (|00>)
 => 1.000000 +0.000000i|0> (1.000000e+00) (|00>)
Uf0 1.000000 +0.000000i|1> (1.000000e+00) (|01>)
 => 1.000000 +0.000000i|1> (1.000000e+00) (|01>)
Uf0 1.000000 +0.000000i|2> (1.000000e+00) (|10>)
 => 1.000000 +0.000000i|2> (1.000000e+00) (|10>)
Uf0 1.000000 +0.000000i|3> (1.000000e+00) (|11>)
 => 1.000000 +0.000000i|3> (1.000000e+00) (|11>)

Uf1 1.000000 +0.000000i|0> (1.000000e+00) (|00>)
 => 1.000000 +0.000000i|0> (1.000000e+00) (|00>)
Uf1 1.000000 +0.000000i|1> (1.000000e+00) (|01>)
 => 1.000000 +0.000000i|1> (1.000000e+00) (|01>)
Uf1 1.000000 +0.000000i|2> (1.000000e+00) (|10>)
 => 1.000000 +0.000000i|3> (1.000000e+00) (|11>)
```

```
Uf1 1.000000 +0.000000i|3> (1.000000e+00) (|11>)
 => 1.000000 +0.000000i|2> (1.000000e+00) (|10>)

Uf2 1.000000 +0.000000i|0> (1.000000e+00) (|00>)
 => 1.000000 +0.000000i|1> (1.000000e+00) (|01>)
Uf2 1.000000 +0.000000i|1> (1.000000e+00) (|01>)
 => 1.000000 +0.000000i|0> (1.000000e+00) (|00>)
Uf2 1.000000 +0.000000i|2> (1.000000e+00) (|10>)
 => 1.000000 +0.000000i|2> (1.000000e+00) (|10>)
Uf2 1.000000 +0.000000i|3> (1.000000e+00) (|11>)
 => 1.000000 +0.000000i|3> (1.000000e+00) (|11>)

Uf3 1.000000 +0.000000i|0> (1.000000e+00) (|00>)
 => 1.000000 +0.000000i|1> (1.000000e+00) (|01>)
Uf3 1.000000 +0.000000i|1> (1.000000e+00) (|01>)
 => 1.000000 +0.000000i|0> (1.000000e+00) (|00>)
Uf3 1.000000 +0.000000i|2> (1.000000e+00) (|10>)
 => 1.000000 +0.000000i|3> (1.000000e+00) (|11>)
Uf3 1.000000 +0.000000i|3> (1.000000e+00) (|11>)
 => 1.000000 +0.000000i|2> (1.000000e+00) (|10>)

====== TEST DZIAŁANIA ROZWIĄZANIA PROBLEMU DEUTSCHA ======

Uf0
-0.707107 +0.000000i|3> (4.999999e-01) (|11>)
 0.707107 +0.000000i|2> (4.999999e-01) (|10>)

Uf1
-0.707107 +0.000000i|1> (4.999999e-01) (|01>)
 0.707107 +0.000000i|0> (4.999999e-01) (|00>)

Uf2
 0.707107 +0.000000i|1> (4.999999e-01) (|01>)
-0.707107 +0.000000i|0> (4.999999e-01) (|00>)

Uf3
 0.707107 +0.000000i|3> (4.999999e-01) (|11>)
-0.707107 +0.000000i|2> (4.999999e-01) (|10>)
```

Wyjście programu składa się z dwóch sekcji.

W pierwszej zweryfikowane zostało prawidłowe działanie przekształceń $\mathbf{U}_{f0}$, $\mathbf{U}_{f1}$, $\mathbf{U}_{f2}$ i $\mathbf{U}_{f3}$ dla wszystkich możliwych wartości wejściowych.

W drugiej sprawdzone zostało rozwiązanie problemu Deutscha. Wynik jest zgodny z oczekiwaniem, dla funkcji f0 i f3, dla których f(0)=f(1) otrzymano na qubicie wejściowym (po lewej) 1, a dla pozostałych funkcji f1 i f2, dla których f(0)≠f(1) uzyskano na nim wartość 0.

Problem Bernsteina-Vaziraniego (ćw. 4)

Kod (lab3-3.c)

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <quantum.h>

void quantum_uf (int a, quantum_reg *reg)
{
  int i;

  for (i = reg->width - 1; i; --i)
    if (a & 1 << (i - 1))
      quantum_cnot(i, 0, reg);
}

void printa(int a, int n)
{
  int i;

  printf("a = ");
  for (i = n - 1; i >= 0; --i)
    printf(a & 1 << i ? "1" : "0");
  printf("\n");
}

void quantum_uf_test(int a, int x, int n)
{
  quantum_reg reg = quantum_new_qureg(x << 1, n + 1);

  printa(a, n);

  printf("Uf");
  quantum_print_qureg(reg);

  quantum_uf(a, &reg);

  printf("=>");
  quantum_print_qureg(reg);

  quantum_delete_qureg(&reg);
}

int main ()
{
  int a, n, m;
  quantum_reg reg;

  srand(time(0));

  printf("=============== TEST DZIAŁANIA FUNKCJI Uf ================\n\n");

  quantum_uf_test(31, 31, 5);
  quantum_uf_test(30, 31, 5);
  quantum_uf_test(31, 30, 5);

  printf("========== ROZWIĄZANIE KLASYCZNE (n wywołań Uf) ==========\n\n");

  a = 1000;
  n = 10;
```

```
  printa(a, n);

  for (m = 0; m < n; ++m)
  {
    reg = quantum_new_qureg(1 << (m + 1), n + 1);

    quantum_uf(a, &reg);

    printf("a%i:", m);
    quantum_print_qureg(reg);

    quantum_delete_qureg(&reg);
  }

  printf("========= ROZWIĄZANIE KWANTOWE (1 wywołanie Uf) ==========\n\n");

  printa(a, n);

  reg = quantum_new_qureg(1, n + 1);

  for (m = 0; m < n + 1; ++m)
    quantum_hadamard(m, &reg);

  quantum_uf(a, &reg);

  for (m = 0; m < n + 1; ++m)
    quantum_hadamard(m, &reg);

  printf("a:");
  quantum_print_qureg(reg);

  quantum_delete_qureg(&reg);

  return 0;
}
```

Wyniki

```
=============== TEST DZIAŁANIA FUNKCJI Uf ================

a = 11111
Uf 1.000000 +0.000000i|62> (1.000000e+00) (|11 1110>)
=> 1.000000 +0.000000i|63> (1.000000e+00) (|11 1111>)

a = 11110
Uf 1.000000 +0.000000i|62> (1.000000e+00) (|11 1110>)
=> 1.000000 +0.000000i|62> (1.000000e+00) (|11 1110>)

a = 11111
Uf 1.000000 +0.000000i|60> (1.000000e+00) (|11 1100>)
=> 1.000000 +0.000000i|60> (1.000000e+00) (|11 1100>)

========== ROZWIĄZANIE KLASYCZNE (n wywołań Uf) ==========

a = 1111101000
a0: 1.000000 +0.000000i|2> (1.000000e+00) (|000 0000 0010>)
```

```
a1: 1.000000 +0.000000i|4> (1.000000e+00) (|000 0000 0100>)
a2: 1.000000 +0.000000i|8> (1.000000e+00) (|000 0000 1000>)
a3: 1.000000 +0.000000i|17> (1.000000e+00) (|000 0001 0001>)
a4: 1.000000 +0.000000i|32> (1.000000e+00) (|000 0010 0000>)
a5: 1.000000 +0.000000i|65> (1.000000e+00) (|000 0100 0001>)
a6: 1.000000 +0.000000i|129> (1.000000e+00) (|000 1000 0001>)
a7: 1.000000 +0.000000i|257> (1.000000e+00) (|001 0000 0001>)
a8: 1.000000 +0.000000i|513> (1.000000e+00) (|010 0000 0001>)
a9: 1.000000 +0.000000i|1025> (1.000000e+00) (|100 0000 0001>)

========== ROZWIĄZANIE KWANTOWE (1 wywołanie Uf) ==========

a = 1111101000
a: 1.000000 +0.000000i|2001> (9.999998e-01) (|111 1101 0001>)
```

Wyjście programu składa się z trzech sekcji.

W pierwszej dla trzech przykładowych zestawów danych (parametru *a* i argumentu funkcji) sprawdzono poprawność działania przekształcenia $\mathbf{U}_f$.

W drugiej rozwiązano problem zakładając działanie na komputerze klasycznym – używając przekształcenia $\mathbf{U}_f$ tyle razy, ile wynosi długość wejścia. Kolejne rejestry wartości parametru *a* odtworzono w kolejnych obliczeniach i można je odczytać z rejestru wyjściowego (skrajnego z prawej) wyniku każdego z pomiaru.

W trzeciej sekcji problem rozwiązano na komputerze kwantowym – używając przekształcenia $\mathbf{U}_f$ tylko raz. Wartość parametru *a* znalazła się w rejestrze wejściowym, czyli wyniku po odrzuceniu skrajnego prawego rejestru.