1 Машины Тьюринга

Работу требуется выполнять в системе turingmachine.io.

Для сдачи заданий 1-2 требуется прикрепить файлы YAML с исходным кодом проекта. Каждый файлы должен иметь наименование задание_пункт.yml, к примеру 1_1.yml для первой задачи первого задания.

1.1 Операции с числами

Реализуйте машины Тьюринга, которые позволяют выполнять следующие операции:

1. Сложение двух унарных чисел (1 балл)
   Алгоритм:
   (a) Движемся вправо по 1, пока не достигнем +
   (b) Когда достигли +, заменяем его на 1 и смещаемся влево
   (c) Движемся влево по 1, пока не достигнем λ, после того как достигли, смещаемся вправо
   (d) Первую встретившуюся 1 заменяем на λ и смещаемся вправо

```
input: '111+11'
blank: ' '
start state: start
table:
start:
    [1]: R
    +  : {write: 1, L: del}
del:
    [1]: L
    ' ': {write: ' ', R: step}

step:
    1 : {write: ' ', R: done}

done:
```

2. Умножение унарных чисел (1 балл)
   Алгоритм:
   (a) Ищем второй множитель, заменяем в нём первую 1 на x
   (b) Если встретили λ, то ставим =
   (c) Идём от знака = до первого множителя и замеянем все 1 на x
   (d) После того, как в первом множителе не осталось 1, возвращаем их на место и также ставим ещё один x во втором множителе

(e) Повторяем эту операцию, пока во втором множителе не будет столько x, сколько там было 1 изначально, как только набралось такое количество, заменяем x на 0.

```
# Adds 1 to a binary number.
input: '111*11'
blank: ' '
start state: v1
table:
v1:
    1: R
    '*' : {R: v2}

v2:
    1: {write: 'x', R: v3}
    '=': {L: v11}
    x: R

v3:
    1: R
    ' ': {write: '=', L: v4}
    '=': {L: v4}

v4:
    1: {L: v5}
    x: {L: v5}

v5:
    1: L
    x: L
    '*' : {L: v6}

v6:
    x: L
    1: {write: 'x', R: v8}
    ' ' : {R: v7}

v7:
    x: {write: '1', R: v7}
    '*': {R: v2}

v8:
    '*': R
    1: R
    x: R
    '=': {R: v9}

v9:
```

```
    1: R
    ' ': {write: '1', L: v10}

v10:
    1: L
    '=': {L: v4}

v11:
    x: {write: '1', L: v11}
    '*': {L: v12}

v12:
```

1.2 Операции с языками и символами

Реализуйте машины Тьюринга, которые позволяют выполнять следующие операции:

1. Принадлежность к языку $L = \{0^n 1^n 2^n\}, n \geq 0$ (0.5 балла)

   (a) Поочерёдно заменяем 0,1 и 2 на временный символ a, затем возвращаемся назад в начало слова и повторяем процесс

   (b) Если при выполнении первого шага с распознаванием нуля мы находим пустой символ, то переходим к состоянию успеха: удаляем строку и печатаем символ s

   (c) Если посреди последовательности из 0, 1 или 2 мы находим другой символ, то переходим в состояние неуспеха: удаляем всю строку и печатаем символ f

```
input: '0011222'
blank: ' '
start state: v0
table:

v0:
    ' ': {L: v4}
    0: {write: a, R: v1}
    [1, 2]: {R: v5}
    a: R

v1:
    [0, a]: R
    1: {write: a, R: v2}
    [2, ' ']: {L: v5}

v2:
    [1, a]: R
    2: {write: a, L: v3}
    [0,' ']: {L: v5}
```

```
v3:
    [a, 2, 1, 0]: L
    ' ': {R: v0}

v4:
    a: {write: ' ', L}
    ' ': {write: s, R: v7}

v5:
    [a, 0, 1, 2]: R
    ' ': {L: v6}

v6:
    [a, 0, 1, 2]: {write: ' ', L}
    ' ': {write: f, R: v7}
v7:
    ' ': L
```

2. Проверка соблюдения правильности скобок в строке (минимум 3 вида скобок) (0.5 балла)

   (a) Находим первую правую скобку

   (b) Затем проверяем предшествующую ей, если на такого же типа, то заменяем обе на временный символ a и переходим в начальное состояние, иначе переходим в состояние неуспеха: удаляем строку и печатаем f.

   (c) Если переходя обратно в начальное состояние мы находим конец строки, то переходим в состояние успеха: печатаем символ s и удаляем остальную строку.

```
input: '([({})])({[]}){}'
blank: ' '
start state: v1
table:
v1:
    ' ': {L: v8}
    a: R
    ['(','[','{']: {R: v2}
    ')': {write: a, L: v3}
    ']': {write: a, L: v4}
    '}': {write: a, L: v5}

v2:
    ' ': {L: v12}
     a: R
    ['(','[','{']: R
    ')': {write: a, L: v3}
    ']': {write: a, L: v4}
    '}': {write: a, L: v5}
```

```
v3:
    a: L
    '(': {write: a, L: v6}
     ['{', '[']: {write: a, L: v12}
    ' ': {R: v12}

v4:
     a: L
    '[': {write: a, L: v6}
    ['{', '(']: {write: a, L: v12}
    ' ': {R: v12}

v5:
    a: L
    '{': {write: a, L: v6}
    ['(', '[']: {write: a, L: v12}
    ' ': {R: v12}

v6:
    ' ': {R: v1}
    [a, '{', '[', '(']: {R: v7}

v7:
    a: {L: v1}
    ' ': L

v8:
    ' ': {R: v9}
    a: L
    ['{', '[', '(']: {L: v6}

v9:
    [a, ' ']: {write: d, R: v10}

v10:
    [a, '{', '[', '(', '}', ']', ')']: R
    ' ': {L: v11}

v11:
    [a, '{', '[', '(', '}', ']', ')']: {write: ' ', L}
    d: L
    ' ': {R: v13}

v12:
    [a, '{', '[', '(', '}', ']', ')', ' ']: {R: v10}
```

```
v13:
    ' ': {write: f, R: v7}
```

3. Поиск минимального по длине слова в строке (слова состоят из символов 1 и 0 и разделены пробелом) (1 балл)

   (a) Если первое слово длиннее, стираем его и восстанавливаем его из временных символов

   (b) Если второе слово оказалось больше, копируем первое на место второго и удаляем то, что осталось от второго

```
input: '1010 010 01 000'
blank: ' '
start state: v1
table:
v1:
    0: {write: a, R: v2}
    1: {write: b, R: v2}
    [a, b]: R
    ' ': {L: v9}
v2:
    [0, 1]: R
    ' ': {R: v3}
v3:
     ' ': {L: v7}
     0: {write: a, L: v5}
     1: {write: b, L: v5}
     [a, b]: {R: v4}
v4:
    [a, b]: R
    0: {write: a, L: v5}
    1: {write: b, L: v5}
    ' ': {L: v22}
v5:
    [a, b]: L
    ' ': {L: v6}
v6:
    [0, 1, a, b]: L
    ' ': {R: v1}
v7:
    ' ': {L: v8}
v8:
    a: {write: 0, L}
    b: {write: 1, L}
    [0, 1]: L
    ' ': {R: done}
```

```
v9:
    [a, b]: L
    ' ': {R: v10}
v10:
    a: {write: 0, R}
    b: {write: 1, R}
    ' ': {R: v11}
v11:
    [a, b, 0, 1]: {write: a, R}
    ' ': {L: v12}
v12:
    a: L
    ' ': {L: v13}
v13:
    [a, b]: L
    0: {write: a, R: v14}
    1: {write: b, R: v17}
    ' ': {R: v20}
v14:
    [a, b]: R
    ' ': {R: v15}
v15:
    a: R
    [0, 1, ' ']: {L: v16}
v16:
    a: {write: 0, L: v12}
    ' ': {L: v12}
v17:
    [a, b]: R
    ' ': {R: v18}
v18:
    a: R
    [0, 1, ' ']: {L: v19}
v19:
    a: {write: 1, L: v12}
    ' ': {L: v12}
v20:
    [a, b]: {write: ' ', R}
    [0, 1]: {L: v6}
    ' ': {R: v21}
v21:
    [a, b]: {write: ' ', R}
    [0, 1]: {L: v6}
    ' ': {R: done}
v22:
    [a, b]: L
```

```
    ' ': {L: v23}
v23:
    [0, 1, a, b]: L
    ' ': {R: v24}
v24:
    [0, 1, a, b]: {write: ' ', R}
    ' ': {R: v25}
v25:
    a: {write: 0, R}
    b: {write: 1, R}
    ' ': {L: v6}
done:
```

2 Квантовые вычисления

Для выполнения заданий по квантовым вычислениям требуется QDK. Его можно скачать здесь: https://docs.microsoft.com/en-us/azure/quantum/install-overview-qdk.

Но можно использовать любой пакет, типа https://qiskit.org/.

В качестве решения задачи надо предоставить схему алгоритма для частного случая при фиксированном количестве кубитов и фиксированных состояниях.

2.1 Генерация суперпозиций 1 (1 балл)

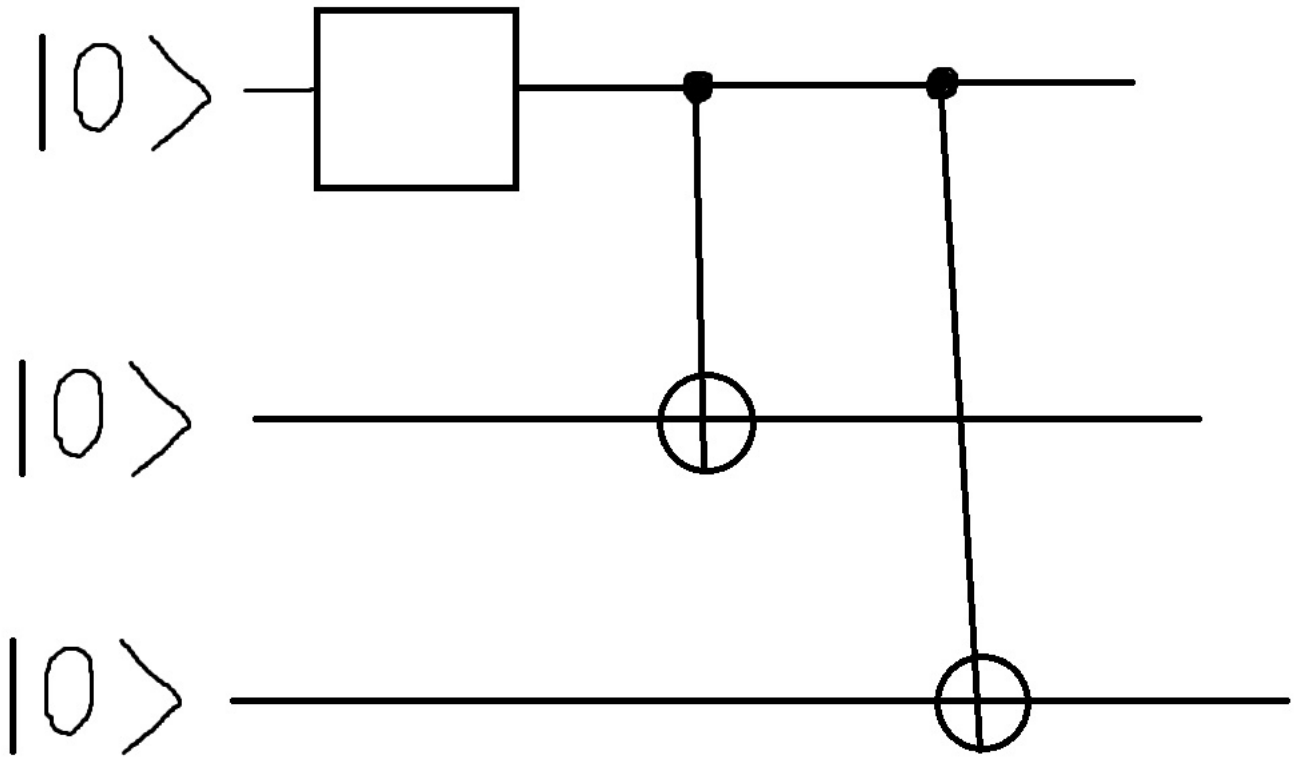
Дано N кубитов ($1 \leq N \leq 8$) в нулевом состоянии $0 \dots 0$. Также дана некоторая последовательность битов, которое задаёт ненулевое базисное состояние размера N. Задача получить суперпозицию нулевого состояния и заданного.

$$S = \frac{1}{\sqrt{2}}(0 \dots 0 + \psi)$$

То есть требуется реализовать операцию, которая принимает на вход:

1. Массив кубитов q_s
2. Массив битов $bits$ описывающих некоторое состояние ψ. Это массив имеет тот же самый размер, что и qs. Первый элемент этого массива равен 1.

1. В начале у нас есть N независимых кубитов |0>
2. Первые кубиты векторов различны, применим оператор Адамара к первому кубиту
3. Все кубиты qs = 0, если кубит bits[i] = 1, то нужно запутать i-ый кубит, а если bits[i] = 0, то не нужны, т.к. кубиты совпадают и равны 0



Заготовка для кода:

```
namespace Solution
 {
    open Microsoft.Quantum.Primitive;
    open Microsoft.Quantum.Canon;

    operation Solve(qs: Qubit[], bits: Bool[]) : ()
    {
        body
        {
            H(qs[0]);
            for (i in 1..Length(qs) - 1)
             {
                if (bits[i])
                 {
                    CNOT(qs[0], qs[i]);
                }
            }
        }
    }
```

```
}
```

2.2 Различение состояний 1 (1 балл)

Дано N кубитов ($1 \leq N \leq 8$), которые могут быть в одном из двух состояний:

$$GHZ = \frac{1}{\sqrt{2}}(0\ldots0 + 1\ldots1)$$

$$W = \frac{1}{\sqrt{N}}(10\ldots00 + 01\ldots00 + \cdots + 00\ldots01)$$

Требуется выполнить необходимые преобразования, чтобы точно различить эти два состояния. Возвращать 0, если первое состояние и 1, если второе.

1. Чтобы измерить состояние системы надо измерить кубиты
2. При N $>$ 1 состояние 1: N нулей, либо N единиц, состояние 2: 1 единица
3. При N $=$ 1 состояния не различать (в обоих состояниях может выпасть вектор, который содержит 1 единицу)

Заготовка для кода:

```
namespace Solution {
    open Microsoft.Quantum.Primitive;
    open Microsoft.Quantum.Canon;
    operation Solve (qs : Qubit[]) : Int
    {
        body
        {
            mutable ones = 0;
            for i in 0..Length(qs) - 1 {
                if (M(qs[i]) == One) {  // measurement
                    set ones += 1;
                }
            }
            if (ones == 1) {
                return 1;
            }
            return 0;
        }
    }
}
```

2.3 Различение состояний 2 (2 балла)

Дано 2 кубита, которые могут быть в одном из двух состояний:

$$S_0 = \frac{1}{2}(00 + 01 + 10 + 11)$$

$$S_1 = \frac{1}{2}(00 - 01 + 10 - 11)$$

$$S_2 = \frac{1}{2}(00 + 01 - 10 - 11)$$

$$S_3 = \frac{1}{2}(00 - 01 - 10 + 11)$$

Требуется выполнить необходимые преобразования, чтобы точно различить эти четыре состояния. Возвращать требуется индекс состояния (от 0 до 3).

Заготовка для кода:

```
namespace Solution {
        open Microsoft.Quantum.Primitive;
        open Microsoft.Quantum.Canon;
        operation Solve (qs : Qubit[]) : Int
        {
            body
            {

                return
            }
        }
}
```

2.4 Написание оракула 1 (2 балла)

Требуется реализовать квантовый оракул на N кубитах ($1 \leq N \leq 8$), который реализует следующую функцию: $f(\boldsymbol{x}) = (\boldsymbol{bx}) \mod 2$, где $\boldsymbol{b} \in \{0, 1\}^N$ вектор битов и $\boldsymbol{x}$ вектор кубитов. Выход функции записать в кубит $\boldsymbol{y}$. Количество кубитов N ($1 \leq N \leq 8$).

Заготовка для кода:

```
namespace Solution {
        open Microsoft.Quantum.Primitive;
        open Microsoft.Quantum.Canon;
        operation Solve (x : Qubit[], y : Qubit, b : Int[]) : ()
        {
            body
            {

            }
```

```
            }
        }
```