А-05-19 Филиппов Евгений

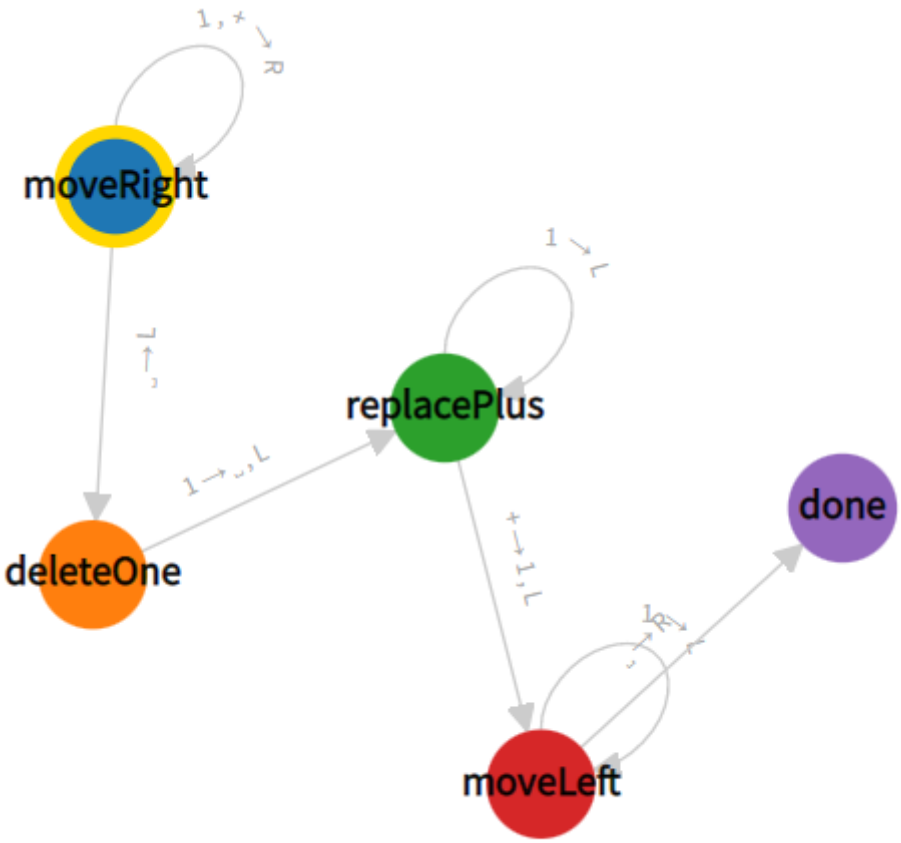
2. Машины Тьюринга

2.1. Операции с числами

Реализуйте машины Тьюринга, которые позволяют выполнять следующие операции:

1. Сложение двух унарных чисел (1 балла)

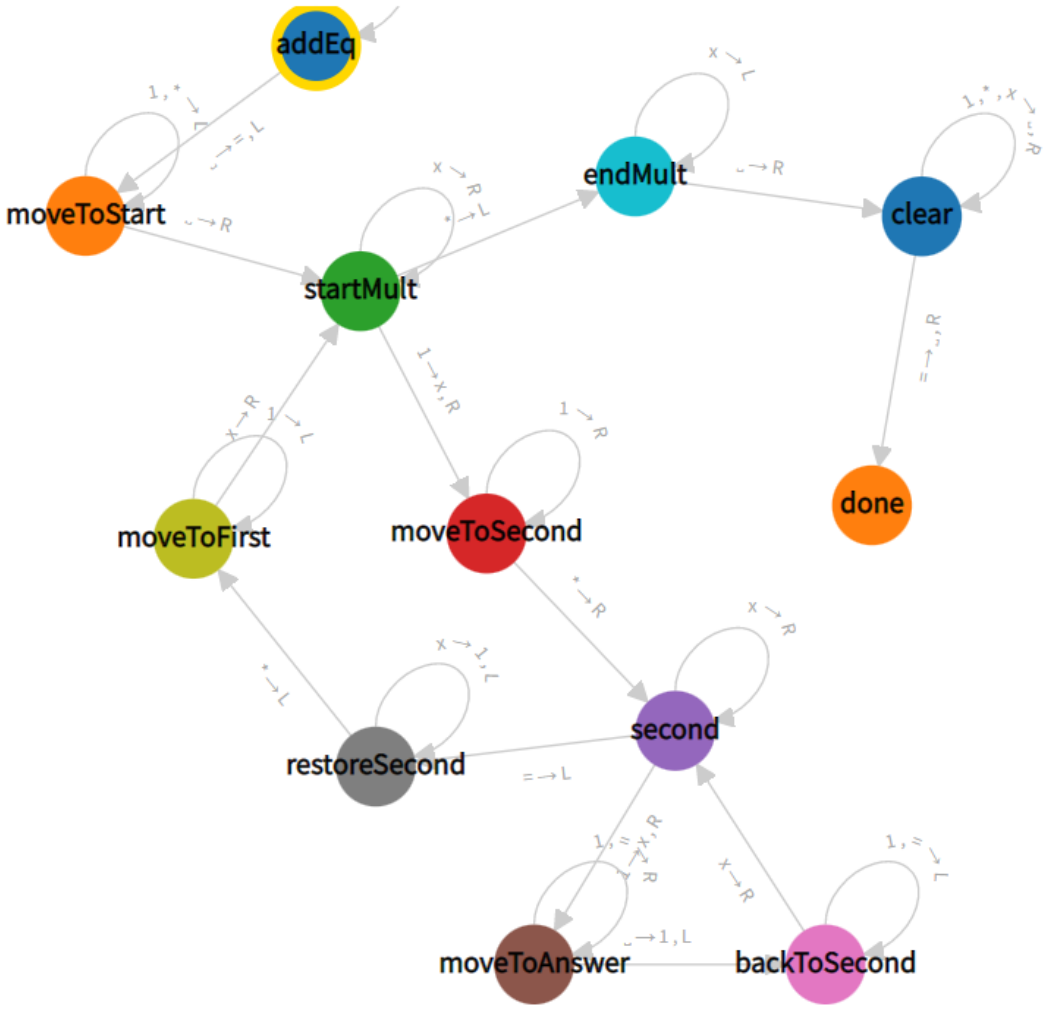
```
#1.1 Сложение двух унарных чисел
input: '111+11'
blank: ' '
start state: moveRight
table:
  # Двигаемся вправо до конца выражения
  moveRight:
    [1,+]: R
    ' ' : {L: deleteOne}
  # Удаляем последнюю единицу
  deleteOne:
    1: {write: ' ', L: replacePlus}
  # Заменяем + на 1
  replacePlus:
    1: L
    +: {write: 1, L: moveLeft}
  # Двигаемся влево к началу выражения
  moveLeft:
    1: L
    ' ': {R: done}
  done:
```



2. Умножение унарных чисел (1 балл)

```
# 1.2 Умножение двух унарных чисел
input: '111*111'
blank: ' '
start state: addEq
table:
  # Двигаемся направо в конец выражения и ставим "="
  addEq:
    [1, '*']: R
    ' ': {write: =, L: moveToStart}
  # Возвращаемся в начало
  moveToStart:
    [1, '*']: L
    ' ': {R: startMult}
  # Начинаем операцию умножения
  startMult:
    1: {write: x, R: moveToSecond}
    x: R
    '*': {L: endMult}
  # Двигаемся направо ко второму множителю
  moveToSecond:
    1: R
    '*': {R: second}
  # Заменяем первую 1 на x
  second:
    x: R
    1: {write: x, R: moveToAnswer}
    =: {L: restoreSecond}
```

```
# Двигаемся направо и переносим эту единицу в ответ
moveToAnswer:
  [1, =]: R
  ' ': {write: 1, L: backToSecond}
# Возвращаемся ко второму множителю
backToSecond:
  [1, =]: L
  x: {R: second}
# Восстанавливаем второй множитель
restoreSecond:
  x: {write: 1, L}
  '*': {L: moveToFirst}
# Переходим к первому числу для обработки остальных разрядов
moveToFirst:
  1: L
  x: {R: startMult}
# Двигаемся к левому краю. Окончание умножения
endMult:
  x: L
  ' ': {R: clear}
# Очищаем все ненужные данные
clear:
  [1, '*', x]: {write: ' ', R: clear}
  =: {write: ' ', R: done}
done:
```

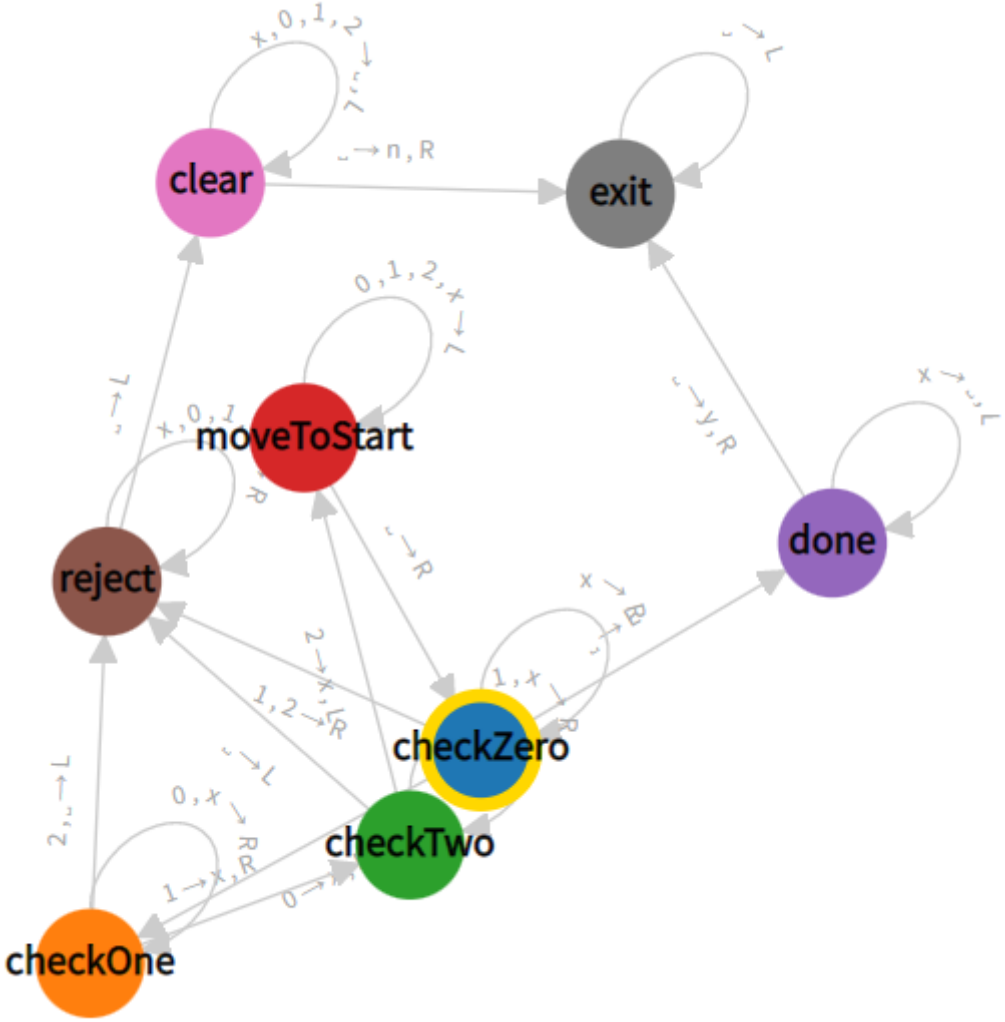


2.2 Операции с языками и символами

Реализуйте машины Тьюринга, которые позволяют выполнять следующие операции:

1. Принадлежность к языку $L = \{0^n1^n2^n\}, n \geq 0$ (0.5 балла)

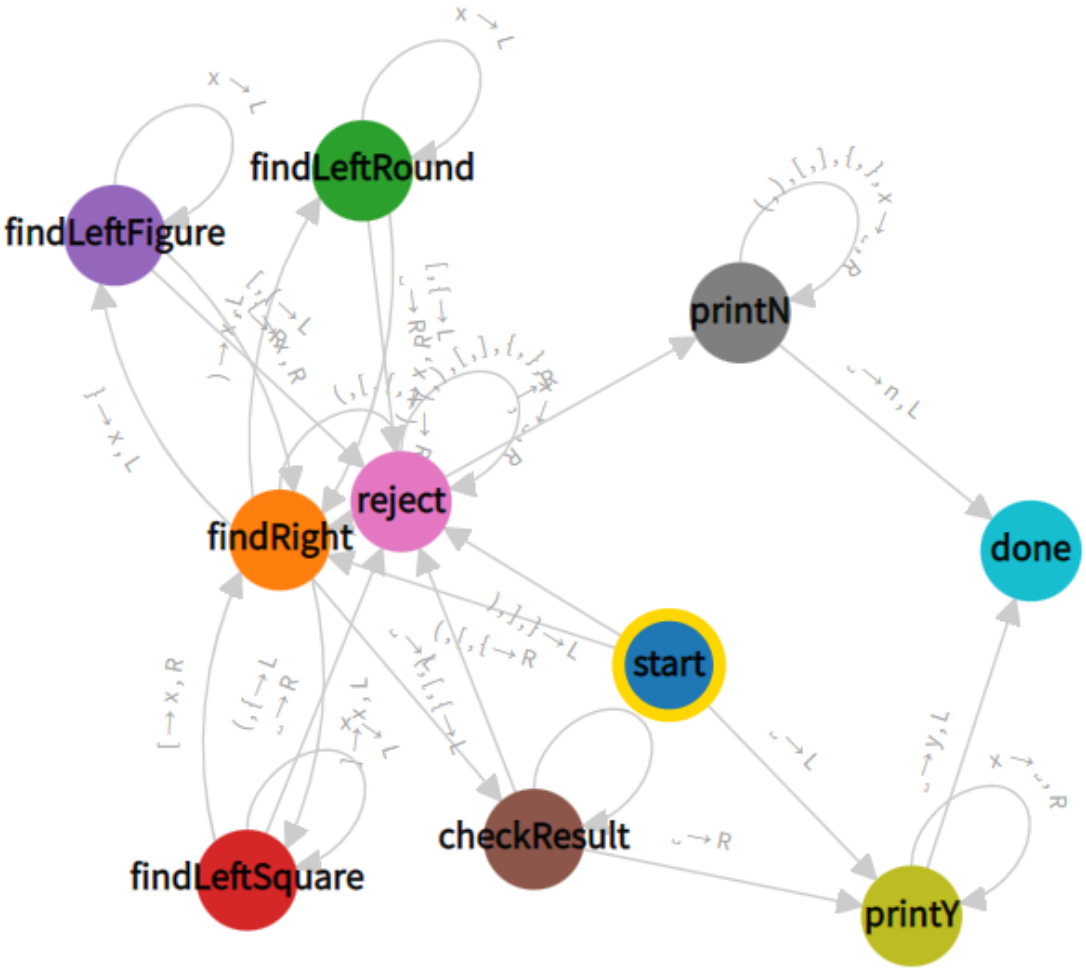
```
# 2.1 Принадлежность к языку L = {0^n1^n2^n}, n>=0
input: '000111222'
blank: ' '
start state: checkZero
table:
  # Ищем 0
  checkZero:
    ' ': {L: done}
    0: {write: x, R: checkOne}
    [1, 2]: {R: reject}
    x: R
  # Ищем 1
  checkOne:
    [0, x]: R
    1: {write: x, R: checkTwo}
    [2, ' ']: {L: reject}
  # Ищем 2
  checkTwo:
    [1, x]: R
    2: {write: x, L: moveToStart}
    ' ': {L: reject}
  # Вернемся в начало
  moveToStart:
    [0, 1, 2, x]: L
    ' ': {R: checkZero}
  # Слово соответствует языку, очищаем ленту и выводим y
  done:
    x: {write: ' ', L}
    ' ': {write: y, R: exit}
  # Слово не соответствует языку,
  reject:
    [x, 0, 1, 2]: R
    ' ': {L: clear}
  # Очищаем ленту и выводим n
  clear:
    [x, 0, 1, 2]: {write: ' ', L}
    ' ': {write: n, R: exit}
  exit:
    ' ': L
```



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

```
input: '([{}])'
blank: ' '
start state: start
table:
  # Старт
  start:
    ' ': {L: printY}
    ['(', '[', '{']: {R: findRight}
    [')', ']', '}']: {L: reject}
  # Ищем правые скобки
  findRight:
    ' ': {L: checkResult}
    ['(', '[', '{', 'x']: R
    ')': {write: 'x', L: findLeftRound}
    ']': {write: 'x', L: findLeftSquare}
    '}': {write: 'x', L: findLeftFigure}
  # Ищем левые круглые скобки
  findLeftRound:
    ' ': {R: reject}
    '(': {write: 'x', R:  findRight}
    ['[', '{']: {L: reject}
    'x': L
  # Ищем левые квадратные скобки
  findLeftSquare:
```

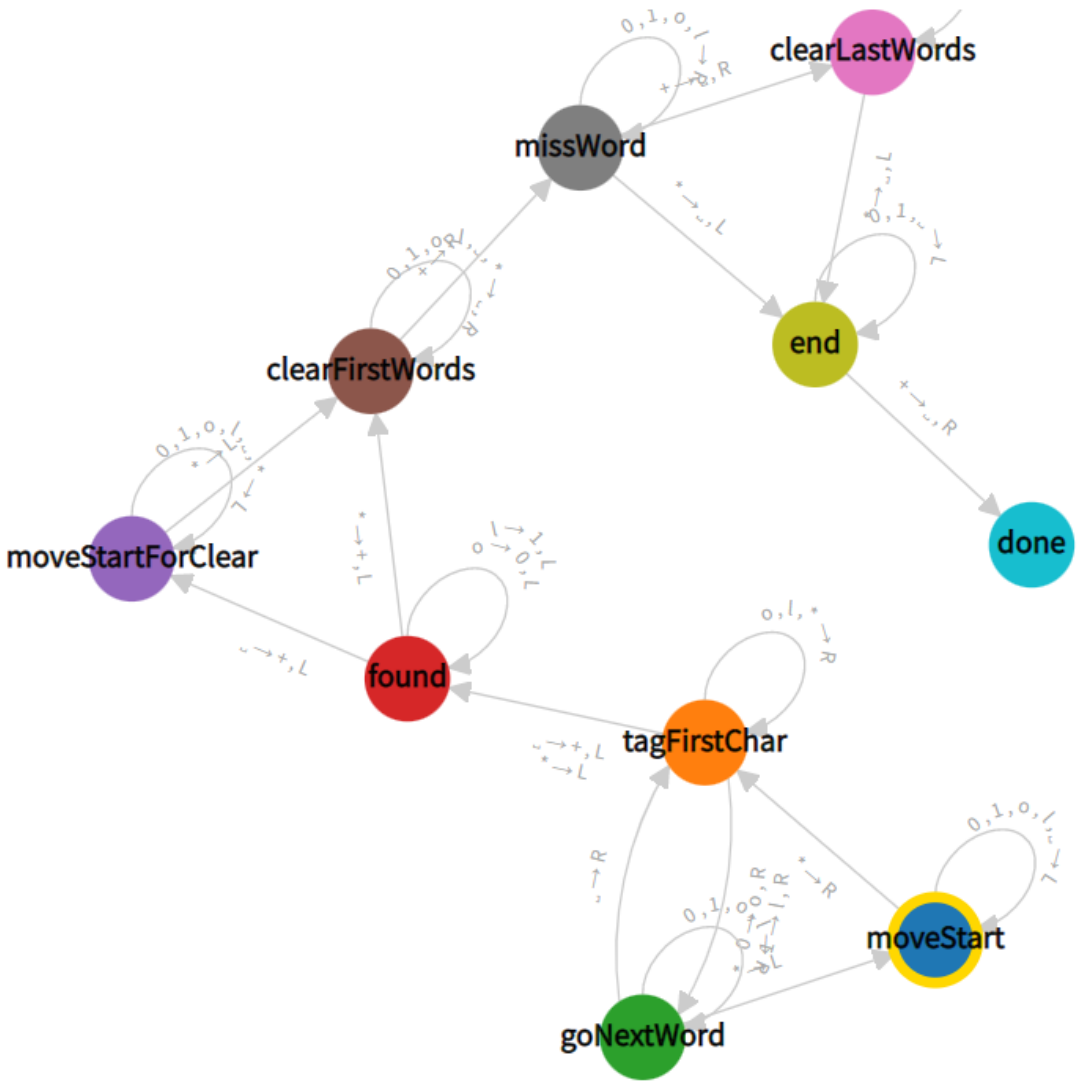
```
  ' ': {R: reject}
  '[': {write: 'x', R: findRight}
  ['(', '{']: {L: reject}
  'x': L
# Ищем левые фигурные скобки
findLeftFigure:
  ' ': {R: reject}
  '{': {write: 'x', R: findRight}
  ['[', '(']: {L: reject}
  'x': L
# Проверяем результат
checkResult:
  ['(', '[', '{']: {L: reject}
  'x': L
  ' ': {R: printY}
# Нашли не закрытую скобку при проверке
reject:
  ['(', ')', '[', ']', '{', '}', 'x']: {write: ' ', R}
  ' ': {R: printN}
# Очищаем ленту и записываем результат n
printN:
  ['(', ')', '[', ']', '{', '}', 'x']: {write: ' ', R: printN}
  ' ': {write: n, L: done}
# Очищаем ленту и записываем результат y
printY:
  ' ': {write: y, L: done}
  'x': {write: ' ', R}

done:
```



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

```
input: '*1111 101 01*'
blank: ' '
start state: moveStart
table:
  # Перемещение в начало строки
  moveStart:
    [0, 1, 'o', 'l', ' ']: L
    '*': {R: tagFirstChar}
  # Помечаем первый неотмеченный символ
  tagFirstChar:
    0: {write: 'o', R: goNextWord}
    1: {write: 'l', R: goNextWord}
    ['o', 'l', '*']: R
    ' ': {write: '+', L: found}
    '*': {L: found}
  # Переходим к следующему слову
  goNextWord:
    [0, 1, 'o','l']: R
    '*': {L: moveStart}
    ' ': {R: tagFirstChar}
  # Слово с минимальной длиной найдено
  found:
    'o': {write: '0', L}
    'l': {write: '1', L}
    ' ': {write: '+', L: moveStartForClear}
```

```
  '*': {write: '+', L: clearFirstWords}
# Перемещаемся в начало строки для очищения ненужных слов
moveStartForClear:
  [0, 1, 'o', 'l', ' ', '*']: L
  '*': {L: clearFirstWords}
# Очищаем ленту до нужного слова
clearFirstWords:
  [0, 1, 'o', 'l', ' ', '*']: {write: ' ', R}
  '+': {R: missWord}
# Очищаем ленту после нужного слова
clearLastWords:
  [0, 1, 'o', 'l', ' ']: {write: ' ', R}
  '*': {write: ' ', L: end}
# Игнорируем нужное нам слово
missWord:
  [0, 1, 'o', 'l']: R
  '+': {write: ' ', R: clearLastWords}
  '*': {write: ' ', L: end}
# Удаляем '+' и устанавливаем каретку на начале нужного слова
end:
  [0, 1, ' ']: L
  '+': {write: ' ', R: done}
done:
```



3. Квантовые вычисления

3.1 Генерация суперпозиций 1 (1 балл)

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

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

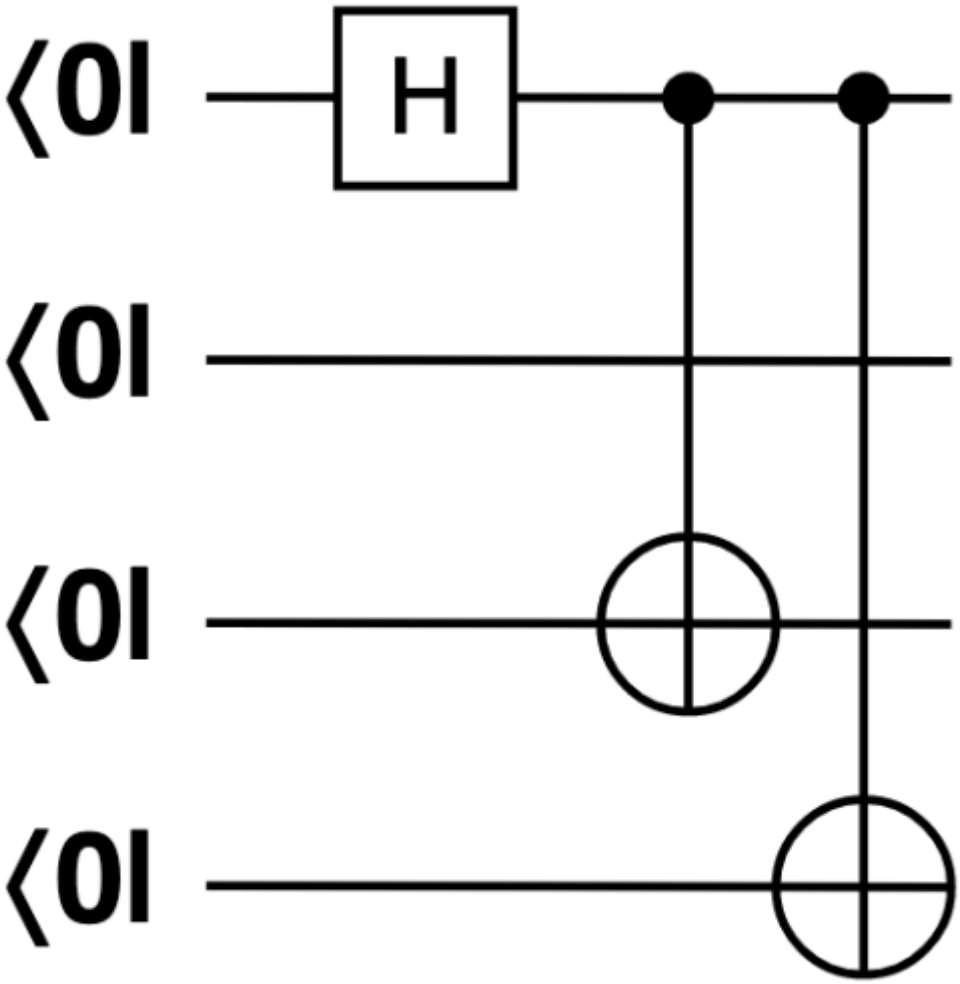
То есть требуется реализовать операцию, которая принимает на вход:

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

Так как первые кубиты векторов различны, то применяем оператор Адамара к первому кубиту

Далее если bits[i] равно 1, то запутываем i-ый кубит (оператор CNOT)

```
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]);
        }
    }
}
```



3.2 Различение состояний 1 (1 балл)

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

$|GHZ\rangle = \frac{1}{\sqrt{2}}(|0\ldots0\rangle + |1\ldots1\rangle)$
$|W\rangle = \frac{1}{\sqrt{N}}(|10\ldots00\rangle + |01\ldots00\rangle + \cdots + |00\ldots01\rangle)$

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

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

```
                return 0;
            }
            return 1;
        }
    }
}
```