Домашняя работа

Щучкин Никита А-13а-19

1. Операции с числами

Постройте машины Тьюринга, которые позволяют выполнять следующие вычисления

1. Сложение двух унарных чисел.

```
input: '11+111'
blank: ' '
start state: right

table:
  right:
    [1,+]: R
    ' ' : {write: =, L: left}
  left:
    [1,+,=]: L
    [' ',0]: {R: change_1}
  change_1:
    [1]: {write: 0, R: update_res}
    [+]: R
    [=]: {L: return_1}
  update_res:
    [1,+,=]: R
    ' ' : {write: 1, L: left}
  return_1:
    [0]: {write: 1, L}
    [+]: L
    ' ': {R: end}
  end:
```

2. Умножение двух унарных чисел.

```
input: '111*11'
blank: ' '
start state: to_end_of_srt

table:
  to_end_of_srt:
    [1, 0, '*']: R
    [' ']: {write: '=', L: to_begin_of_str}
  to_begin_of_str:
    [1, 0, '=', '*']: L
    [' ']: {R: right}
  right:
    [1]: R
    ['*']: {R: second_mult}
  second_mult:
```

```
    [1]: {write: 0, R: check_next}
  check_next:
    [1, 0, '=']: L
    ['*']: {L: first_mult}
  first_mult:
    [1]: {write: 0, R: go_to_end}
    [0]: L
    [' ']: {R: return_1}
  go_to_end:
    [1, 0, '=', '*']: R
    [' ']: {write: 1, L: check_next}
  return_1:
    [0]: {write: 1, R}
    ['*']: {R: search_next_1}
  search_next_1:
    [0]: R
    [1]: {write: 0, R: check_next}
    ['=']: {L: return_1_in_second_mult}
  return_1_in_second_mult:
    [0]: {write: 1, L}
    ['*']: {R: end}

  end:
```

2. Операции с языками и символами

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

   В результате на ленте окажется слово "DA", если слово принадлежит языку, и слово "NET" - в противном случае.

```
input: '001122'
blank: ' '
start state: first

table:
  first:
    [0]: {write: '*', R: second}
    [1,2]: {R: Net}
    ['*']: R
    [' ']: {R: Da}
  second:
    [0]: R
    ['*']: R
    [1]: {write: '*', R: third}
    [2]: {R: Net}
  third:
    ['*']: R
    [0]: {R: Net}
    [' ']: {L: Net}
    [1]: R
    [2]: {write: '*', R: fouth}
  fouth:
    [0,1]: {R: Net}
```

```
    [2]: {L: go_to_begin}
    [' ']: {L: check}
  go_to_begin:
    [0,1,2,'*']: L
    [' ']: {R: first}
  check:
    ['*']: L
    [0,1,2]: {L: Net}
    [' ']: {R: Da}

  Net:
    [0,1,2,'*']: R
    [' ']: {write: N, R: Net2}
  Net2:
    [' ']: {write: e, R: Net3}
  Net3:
    [' ']: {write: t, L: clean}
  Da:
    [0,1,2,'*']: R
    [' ']: {write: D, R: Da2}
  Da2:
    [' ']: {write: a, L: clean}

  clean:
    [D,a,N,e,t]: L
    [0,1,2,'*']: {write: ' ', L}
    [' ']: {R: end}

  end:
```

2. Проверка соблюдения правильности скобок в строке (минимум 3 вида скобок). В результате на ленте окажется слово "YES", если имеем правильную скобочную последовательность, и слово "NO" - в противном случае.

```
input: '{()}[]'
blank: ' '
start state: first

table:
  first:
    ['{','(', '[','x']: R
    ['}']: {write: 'x', L: check_open_1}
    [')']: {write: 'x', L: check_open_2}
    [']']: {write: 'x', L: check_open_3}
    [' ']: {L: check}
  check_open_1:
    ['{']: {write: 'x', L: go_to_left}
    ['x']: L
    ['(', '[',' ']: {L: wrong}
  check_open_2:
    ['(']: {write: 'x', L: go_to_left}
    ['x']: L
    ['{', '[',' ']: {L: wrong}
```

```
  check_open_3:
    ['[']: {write: 'x', L: go_to_left}
    ['x']: L
    ['{', '(',' ']: {L: wrong}
  go_to_left:
    ['{','(','[','}',')',']','x']: L
    [' ']: {R: first}
  check:
    ['x']: L
    ['{','(','[','}',')',']']: {R: wrong}
    [' ']: {R: correct}
  wrong:
    ['{','(','[','}',')',']','x']: L
    [' ']: {R: clean_and_No}
  clean_and_No:
    ['{','(','[','}',')',']','x']: {write: ' ', R}
    [' ']: {write: N, R: No}
  correct:
    ['{','(','[','}',')',']','x']: L
    [' ']: {R: clean_and_Yes}
  clean_and_Yes:
    ['{','(','[','}',')',']','x']: {write: ' ', R}
    [' ']: {write: Y, R: Yes}
  No:
    [' ']: {write: o, R: end}
  Yes:
    [' ']: {write: e, R: S}
  S:
    [' ']: {write: s, R: end}
  end:
```

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

```
input: '10 100 1'
blank: '-'
start state: first

table:
  first:
    ['x','y']: R
    [1]: {write: 'x', R: search_next_word}
    [0]: {write: 'y', R: search_next_word}
    [' ','-']: {write: '<',R: go_to_right}
  search_next_word:
    [0,1]: R
    [' ']: {R: first}
    ['-']: {L: go_to_left}
  go_to_left:
    [0,1,'x','y',' ']: L
    ['-']: {R: first}
  go_to_right:
    [0,1,'x','y',' ']: R
```

```
    ['-']: {L: recovery}
  recovery:
    [0,1,' ','x','y']: {write: '-', L}
    ['<']: {write: '-', L: go_to_space}
  go_to_space:
    ['x']: {write: 1, L}
    ['y']: {write: 0, L}
    [' ']: {write: '-', L: clear}
  clear:
    [0,1,'x','y',' ']: {write: '-', L}
    ['-']: {R: go_to_begin_of_word}
  go_to_begin_of_word:
    ['-']: R
    [0,1]: {L: end}
  end:
```

3. Квантовые вычисления

3.1 Генерация суперпозиций 1

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

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

То есть требуется реализовать операцию, которая принимает на вход:

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

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

3.3 Различение состояний 2

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

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

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

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

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

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

```
operation Solve(qs : Qubit[]) : Int
{
  body
  {
    H(qs[0]);
    H(qs[1]);
    if (M(qs[0]) == Zero)
    {
      if (M(qs[1]) == Zero)
      {
        return 0;
      }
      else
      {
        return 1;
      }
    }
    else
    {
      if (M(qs[1]) == Zero)
      {
        return 2;
      }
      else
      {
        return 3;
      }
    }
  }
}
```