Поиск состояний, создание информационного дерева

Введем константу **dependency_step** - вычисляется как наибольшая величина задержки:

$$\text{dependency\_step} = \max_{\forall i \in \{0,\ldots,t-1\}} (t-i) \in y(t)$$

Полученное значение будет в дальнейшем использоваться как *минимальное количество уровней - 1* , которое необходимо проверить у двух вершин для проверки изоморфности поддеревьев.

Также введем счетчик для нумерации состояний:

$$\text{state\_num} = 1$$

Алгоритм

Шаг 0.0

`ready_states = []`

`states_in_process = [v0]` , где `v0` - корень дерева.

`state_num = 1`

`const dependency_step = 1`

`name_map = { v0 : [1] }`

`queue = []`

`checked_marks = []`

Шаг 0.1

1. Добавляем в очередь детей вершины `v0` .
2. Переходим на **шаг 1**.

Шаг 1

Проверяем `states_in_process` .

1. Если в нем есть вершины, переходим на **шаг 3**.
2. Если вершин нет, переходим на **шаг 2**.

Шаг 2 (halt)

1. Вывести результат.
2. Завершение работы программы.

Шаг 3

`checked_marks = []`

`DFS(T)` - обход в глубину дерева `T` .

`BFS(T)` - обход в ширину дерева `T` .

Если в дереве нет первой вершины `x` из `queue` , тогда добавляем `x` в дерево.

Строим `dependency_step + 1` уровень для `x` и первой вершины `y` из `dfs_list = DFS(T)` .

1. Если в `checked_marks` есть `y` , тогда берем следующий элемент и переходим в (1), в противном случае переходим в (2).
2. Если `BFS(x) == BFS(y)` , тогда `name_map[x] = f"{name_map[y]}"` **шаг 4** (пометили изоморфную вершину `x` той же меткой, что и у `y`).
3. Если `BFS(x) != BFS(y)` и `dfs_list` не пустой, тогда кладем `y` в `checked_marks` , берем следующую вершину `y` из `dfs_list` , и возвращаемся в (1).
4. Если `BFS(x) != BFS(y)` и в `dfs_list` остался только `x` , тогда
    1. `name_map[x] = f"{++state_num}"` (добавили новое состояние).
    2. добавляем `x` в `states_in_process`
    3. добавляем детей `x` в `queue` .
    4. Переходим на **шаг 4**.

Шаг 4

1. Проверяем каждую вершину `x` из `states_in_process` , все ли дети у нее существуют и если да - удаляем `x` из `states_in_process` и добавляем `x` в `ready_states` .
2. Переходим на **шаг 1**.

Пример работы алгоритма

4. Построить диаграмму Мура, каноническую таблицу и канонические уравнения для функции

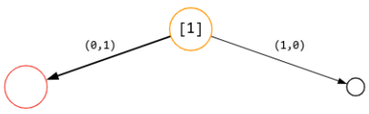
$$y(t) = \begin{cases} \overline{x}(1) & \text{при } t = 1, \\ y(t-1) \oplus x(t) & \text{при } t \geq 2. \end{cases}$$

В данном случае $\mathrm{dependency\_step} = 1$, что соответствует выражению $y(t-1) \oplus x(t)$.

вершиной [1] называю начальную. Остальные по полному имени

Итерация 1 (не в смысле программы)

dependency_step=1 шаг: 0,1,2, [3]

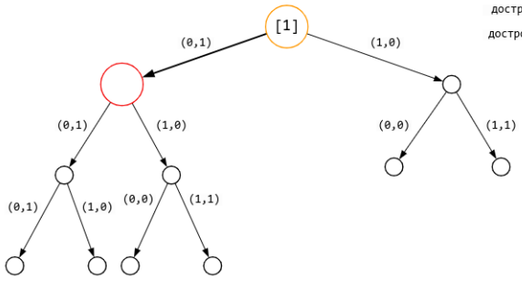


добавим в очередь детей новой вершины [1]

queue: [(0),(1)] states_in_process: [[1]] <- здесь лежат вершины
vertices: [[1]] ready_states: [ ]

начинаем сравнивать вершину (0,) с первой вершиной по обходу вглубину - [1]
вершины (0,) нет в графе. Добавим в граф
vertices: [[1], (0,)]

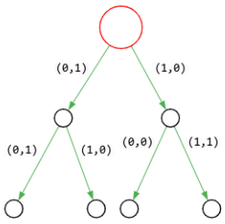
достроим [1] (dependency_step + 1) уровень

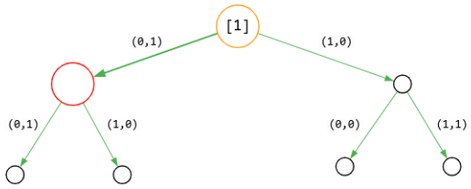


достроим (0,) (dependency_step + 1) уровень

ready_states: [ ] states_in_process: [[1]]
queue: [
(1), - (0) вышла
]
vertices: [[1], (0,)]

сравним обходом в ширину





[(0,1), (1,0), (0,1), (1,0), (0,0), (1,1)]

[(0,1), (1,0), (0,1), (1,0), (0,0), (1,1)]

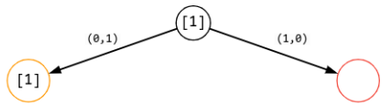
деревья [1] и (0) изоморфны

помечаем (0) как [1]
начинаем сравнивать вершину (1,) с вершиной (0,)
вершины (1,) нет в графе. Добавим в граф

queue: [ ]

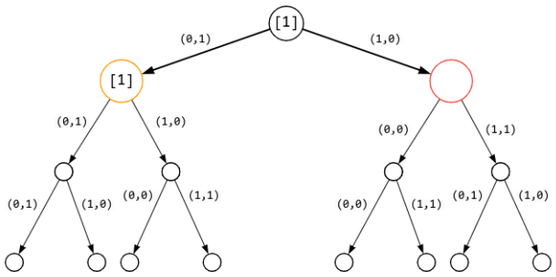
vertices: [[1], (0,), (1,)]
ready_states: [ ]
states_in_process: [[1]]

Итерация 2

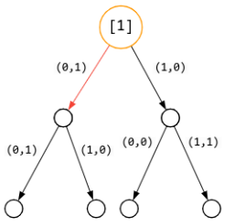


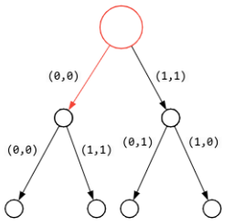
достроим (1,) второй уровень, первый из кеша

достроим (0,) второй уровень, первый из кеша

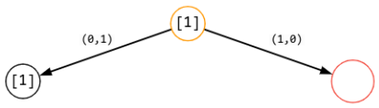


сравним обходом в ширину





не изоморфны. Идем далее
начинаем сравнивать вершину (1,) с вершиной [1]



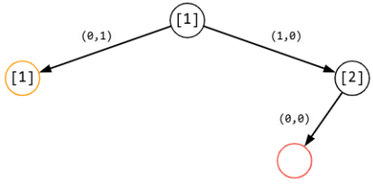
[1] имеет ту же метку, что и вершина (0,)
в графе нет изоморфных поддеревьев данному
помечаем (1,) как [2] - добавляем детей [2] в очередь
queue: [
(1,0), (1,1)
]

после добавления вершины проверяем метки у детей states_in_process
ready_states: [ ]
states_in_process: [[1]]
видим, что у [1] есть все дети, перемещаем в ready_states

добавляем (1,) в states_in_progress
ready_states: [[1]]
states_in_process: [(1,)]

начинаем сравнивать вершину (1,0) с вершиной (0,)

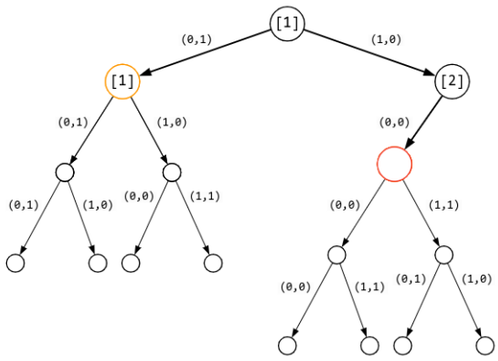
Итерация 3



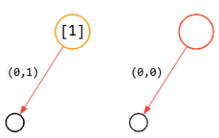
queue: [
(1,1)
]

вершины (1,0) нет в графе - добавляем в граф
vertices: [[1], (0,), (1,), (1,0)]

достроим для (0,0) два уровня и для (0,) два уровня (далее не буду говорить о кеше)

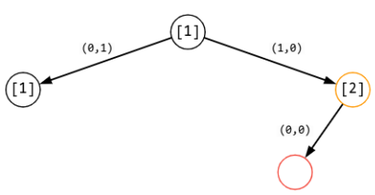


сравним обходом в ширину

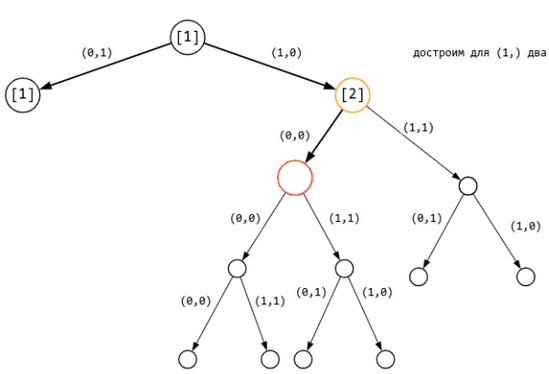


не изоморфны. Идем далее
[1] имеет ту же метку, что и вершина (0,). Идем далее по глубине

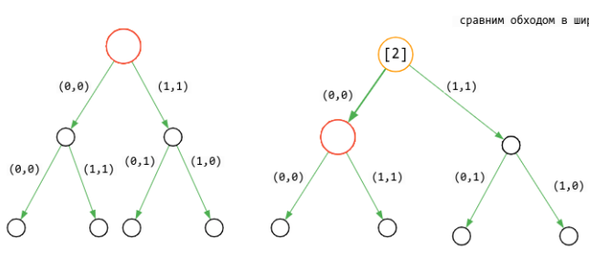
начинаем сравнивать вершину (1,0) с вершиной (1,)



достроим для (1,) два уровня и для (1,0) два уровня

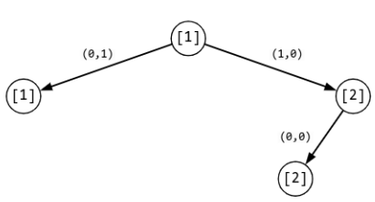


сравним обходом в ширину



итерация 4

деревья (1,0) и (1,) изоморфны
помечаем (1,0) меткой (1,), т.е. [2]

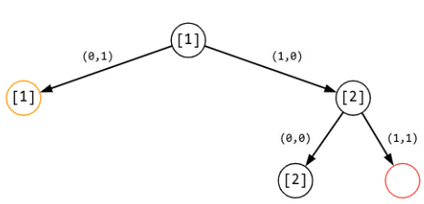


queue: [
(1,1)
]

ready_states: [[1]]

после добавления вершины проверяем метки у детей вершин states_in_process
states_in_process: [(1,)]

начинаем сравнивать второго ребенка первой вершины state_in_process (1,), то есть (1,1) с вершиной (0,)



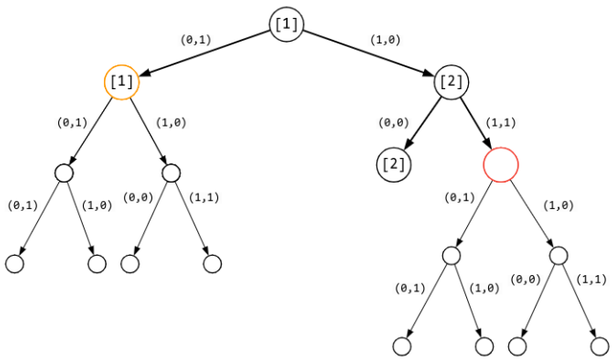
queue: [ ]

вершины (1,1) нет в графе - добавляем в граф

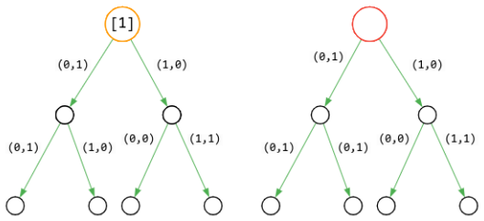
vertices: [[1], (0,), (1,), (1,0), (1,1)]

достроим (1,1) два уровня и для (0,) два уровня

достроим (1,1) два уровня и для (0,) два уровня



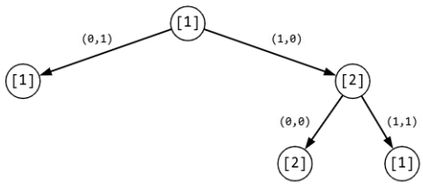
сравним обходом в ширину



деревья (1,1) и (0,) изоморфны
помечаем (1,1) меткой (0,), т.е. [1]

итерация 5

деревья (1,1) и (0,) изоморфны
помечаем (1,1) меткой (0,), т.е. [1]

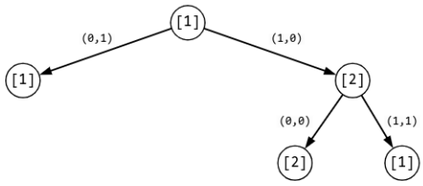


queue: [ ]
ready_states: [[1]]
states_in_process: [(1,)]
после добавления вершины проверяем метки у детей вершин states_in_process
замечаем, что у (1,) есть оба ребенка, значит надо перенести его в ready_states

ready_states: [[1], (1,)]
states_in_process: [ ]

vertices: [[1], (0,), (1,), (1,0), (1,1)]

states_in_process пуста, значит можно прекращать работу программы



Текстовая версия

```
Шаг 0.0
name_map = { () : [1] }
states_in_process = [()]
ready_states = []
checked_marks = []
queue = []
state_num = 1
T: ()


Шаг 0.1
queue = [(0,), (1,)]
T: ()


Шаг 1
Есть вершина ()


Шаг 3
checked_marks = []
Добавляем (0,) в дерево
queue = [(1,)]

T:   ()
 (0,)

bfs_list = BFS(T) = [ (0,), () ]

Строим 2 уровня для (0,)
Строим 2 уровня для ()
```

```
BFS( () ) == BFS( (0,) )

name_map = {
    () : [1],
    (0,) : name_map[ () ]
}


Шаг 4
в states_in_process = [()] для () не существует (1,)
не перемещаем в ready_states


Шаг 1
Есть вершина ()


Шаг 3
checked_marks = []
Добавляем (1,) в дерево
queue = [ ]

bfs_list = [ (0,), (), (1,) ]

T:   ()
 (0,)  (1,)

Строим 2 уровня для (1,)
Строим 2 уровня для (0,)

BFS( (0,) ) != BFS( (1,) )
checked_marks = [ name_map[(0,)] ]

bfs_list = [ (), (1,) ]
Проверяем checked_marks и видим там [1]

bfs_list = [ (1,) ]

(1,) == (1,) # шаг3 пункт 4: x == dfs_list[0]

name_map = {
    () : [1],
    (0,) : [1],
    (1,) : [++state_num],
}

state_num = 2

добавялем детей (1,) в queue
queue = [(1,0), (1,1)]

добавляем (1,) в states_in_process
states_in_process = [ (), (1,) ]


Шаг 4.
проверим детей states_in_process = [ (), (1,) ]:
у вершины () все дети инициализированы, что видно
на диаграмме T:   ()
              (0,)  (1,)
Перемещаем () в ready_states:
ready_states = [ () ]

Удаляем ее из states_in_process:
states_in_process = [ (1,) ]


Шаг 1
Есть вершина (1,)


Шаг 3
checked_marks = [] # queue = [(1,0), (1,1)]
Добавляем (1,0) в дерево
queue = [(1,1)]
```

```
T:    ()
 (0,)    (1,)
       (1,0)

bfs_list = [ (0,), (), (1,), (1,0) ]

Строим 2 уровня для (1,0)
Строим 2 уровня для (0,)

BFS( (1,0) ) != BFS( (0,) )
checked_marks = [ "[1]" ]

bfs_list = [ (), (1,), (1,0) ]
Проверяем checked_marks и видим там [1]

bfs_list = [ (1,), (1,0) ]

BFS( (1,) ) == BFS( (1,0) )

name_map = {
    () : [1],
    (0,) : [1],
    (1,) : [2],
    (1,0) : [2],
}


Шаг 4
проверим детей states_in_process = [ (1,) ]:
Имеем дерево: T:    ()
                (0,)    (1,)
                      (1,0)
не хватает (1,1)


Шаг 1
Есть (1,)


Шаг 3
checked_marks = [] # queue = [(1,1)]
Добавляем (1,1) в дерево
queue = [ ]

T:     ()
 (0,)      (1,)
       (1,0)  (1,1)

bfs_list = [ (0,), (), (1,), (1,0), (1,1)]

Строим 2 уровня для (1,1)
Строим 2 уровня для (0,)

BFS( (1,1) ) == BFS( (0,) )

name_map = {
    () : [1],
    (0,) : [1],
    (1,) : [2],
    (1,0) : [2],
    (1,1) : name_map[(0,)] # [1]
}


Шаг 4
проверим детей states_in_process = [ (1,) ]:
T:     ()
 (0,)      (1,)
       (1,0)  (1,1)

# Так как в дерево добавляются лишь вершины с метками,
# получаем, что все дети уже будут иметь номер какого-то стейта

переместим (1,) в ready_states:
ready_states = [ (), (1,) ]
states_in_process = [ ]
```

```
Шаг 1
Проверяем states_in_process:
states_in_process = [ ]


Шаг 2
T:     ()
 (0,)      (1,)
       (1,0)  (1,1)

ready_states = [ (), (1,) ]

name_map = {
    () : [1],
    (0,) : [1],
    (1,) : [2],
    (1,0) : [2],
    (1,1) : name_map[(0,)] # [1]
}

# если отобразить ready_states через name_map получим:
Q = [ [1], [2] ] - множество состояний
```

Author: Эрик Шелбогашев