Aufgabe 1: Arukone

Niels Heiden
Team-ID: 00380
Team-Name: CTRL-C / CTRL-V

20. November 2023

Inhaltsverzeichnis

1 Lösungsidee

Die Interpretation einer Nandu-Konstruktion erfordert es, den Output jeder Bausteinreihe als Input für die nachfolgende Reihe zu nutzen, bis das Ende erreicht wird. Dieser sequenzielle Prozess wird für jede mögliche Anfangskombination der Lichter durchgeführt, was aufgrund der dualen Natur der Lichtzustände (an/aus) zu einer exponentiellen Anzahl von Kombinationen führt.

Die Laufzeit des Algorithmus beträgt $O(2^{lightcount})$, da er jede dieser Startkonfigurationen durchgehen muss, um die Auswirkungen auf die Endzustände der LEDs zu ermitteln. Diese exponentielle Laufzeit entsteht, weil für jedes Licht, das hinzugefügt wird, die Anzahl der zu testenden Zustände sich verdoppelt. In der Praxis kann dies bedeuten, dass die Laufzeit mit jeder zusätzlichen Lichtquelle rapide ansteigt, was insbesondere bei einer größeren Anzahl von Lichtern zu einer Herausforderung werden kann.

2 Umsetzung

```python
def read_construction_file(filename): #read the construction file and return the number of rows, number of columns, the construction, and the number of lights
    lightcount = 0 #number of lights
    with open(filename, 'r') as f: #open the file
        n, m = map(int, f.readline().strip().split()) #read the first line and get the number of rows and columns
        construction = [list(f.readline().strip().split()) for _ in range(m)] #read the construction

        for i in construction[0]: #for each item in the first row of the construction
            if i.startswith('Q'): #if the item starts with Q
                lightcount += 1 #increment the number of lights
    return n, m, construction, lightcount #return the number of rows, number of columns, the construction, and the number of lights
```

Diese Funktion ist für das einlesen der Konstruktion verantwortlich. Die Konstruktion selbst wird als Liste von Listen eingelesen, wobei jede Liste eine Zeile der Konstruktion repräsentiert. Die erste Zeile wird speziell analysiert, um die Anzahl der Lichter zu zählen, die in der späteren Verarbeitung eine Rolle spielen.

```python
def interpretConstruction(n, m, construction, startingLight, lightcount): # interpret the construction and return the final state of the LEDs
    lightrow = ['O'] * n #create a list of length n with all items set to O
    for i in range(len(construction[0])): #for each item in the first row of the construction
        if construction[0][i].startswith('Q'): #if the item starts with Q
            idx = int(construction[0][i][-1]) - 1 #get the index of the light
            if 0 <= idx < lightcount: #if the index is valid
                lightrow[i] = '1' if startingLight[idx] == '1' else 'O' # set the item at the index to 1 if the light is on or O if the light is off
        if construction[0][i].startswith('X'): #if the item starts with X
            lightrow[i] = 'O' #set the item to O
    startrow = lightrow.copy() #create a copy of the lightrow
```

```
    for i in construction[1:-1]: #for each row in the construction except the first and last
        j = 0 #set j to 0
        while j in range(len(i)): #while j is in range of the length of the row
            if j < len(i) - 1: #if j is less than the length of the row minus 1
                if i[j] == 'R' and i[j+1] == 'r': #if the item at j is R and the item at j+1 is r logic for Red Left Input
                    if lightrow[j] == 'l':
                        lightrow[j] = 'O'
                        lightrow[j+1] = 'O'
                    else:
                        lightrow[j] = 'l'
                        lightrow[j+1] = 'l'
                if i[j] == 'r' and i[j+1] == 'R': #if the item at j is r and the item at j+1 is R logic for Red Right Input
                    if lightrow[j+1] == 'l':
                        lightrow[j] = 'O'
                        lightrow[j+1] = 'O'
                    else:
                        lightrow[j] = 'l'
                        lightrow[j+1] = 'l'
            if i[j] == 'W' and i[j+1] == 'W': #if the item at j is W and the item at j+1 is W logic for White
                if lightrow[j] == 'O' and lightrow[j+1] == 'O':
                    lightrow[j] = 'l'
                    lightrow[j+1] = 'l'
                elif lightrow[j] == 'l' and lightrow[j+1] == 'l':
                    lightrow[j] = 'O'
                    lightrow[j+1] = 'O'
                else: #logic for Blue
                    lightrow[j] = 'l'
                    lightrow[j+1] = 'l'
                j += 1
            if i[j] == 'X': #if the item at j is X
                lightrow[j] = 'O'
            j += 1
        #print(lightrow)
    final_leds = ['1' if lightrow[led_item] == 'l' and construction[-1][led_item].startswith("L") else '0' if construction[-1][led_item].startswith("L") else '' for led_item in range(len(lightrow))] #create a list of length n with all items set to L if the light is on or X if the light is off
    return final_leds
```

Dies ist die Logik der Blöcke und die Simulation der Konstruktion.
Initialisierung des Lichtzustands:
lightrow wird als Liste initialisiert, die so viele Elemente ’O’ (ausgeschaltet) enthält, wie es Spalten n im Rätsel gibt.
Einlesen des Startzustands:
Die Funktion geht durch die erste Zeile der Konstruktion (construction[0]). Für jedes Element, das mit ’Q’ beginnt, wird der entsprechende Index ermittelt, und abhängig von starting-Light(eine Binärzahl welche die verschiedenen Lichtkombinationen repräsentiert), wird das Licht auf ’l’ (eingeschaltet) oder ’O’ (ausgeschaltet) gesetzt.
Durchlaufen der Konstruktion:

Die Funktion iteriert dann durch alle Zeilen der Konstruktion mit Ausnahme der ersten und letzten. Innerhalb jeder Zeile wird jedes Element durchlaufen und Regeln angewendet, die definieren, wie sich die Lichter basierend auf ihren Nachbarn ändern:
Bestimmen des Endzustands:
Nachdem alle Regeln angewendet wurden, durchläuft die Funktion lightrow ein letztes Mal und überprüft mit der letzten Zeile der Konstruktion, um den finalen Zustand jedes LEDs zu bestimmen. Wenn eine LED (markiert mit "L") eingeschaltet ist ('l'), wird sie als '1' gekennzeichnet, ist sie ausgeschaltet ('O'), wird sie als '0' gekennzeichnet.
Rückgabe:
Die Funktion gibt final_leds zurück, eine Liste, die den Endzustand jeder LED repräsentiert, wobei '1' für eingeschaltet und '0' für ausgeschaltet steht.

```
def test_all_flashlight_combinations(n, m, construction, lightcount): #test all flashlight combinations
    print("Testing all flashlight combinations: ") #print the message
    print("Start" + " -> " + "End") #print direction
    for i in range(2**lightcount): #for each flashlight combination
        flashlights = bin(i)[2:].zfill(lightcount) #get the binary representation of the number and pad it with 0s to the length of the number of lights
        final_leds = interpretConstruction(n, m, construction, flashlights, lightcount) #interpret the construction
        print(flashlights + " -> " +  "".join(final_leds)) #print the flashlight combination and the final state of the LEDs
```

Diese Funktion durchläuft alle Kombinationen von Lichtzuständen $2^{lightcount}$ und erzeugt ihre Binäre darstellung $(bin(i)[2:])$ und fügt führende nullen hinzu um die leeren felder zu füllen.

2.1 Beispiele

$nandu1.txt$ Lösung:
Start -> End
00 -> 11
01 -> 11
10 -> 11
11 -> 00
Zeit: 0.0010s
$nandu2.txt$ Lösung:
Start -> End
00 -> 01
01 -> 01
10 -> 01
11 -> 10
Zeit: 0.0015s
$nandu3.txt$ Lösung:
Start -> End
000 -> 1001
001 -> 1000
010 -> 1011
011 -> 1010
100 -> 0101

101 -> 0100
110 -> 0111
111 -> 0110
Zeit: 0.0021s
$nandu4.txt$ Lösung:
Start -> End
0000 -> 00
0001 -> 00
0010 -> 01
0011 -> 00
0100 -> 10
0101 -> 10
0110 -> 11
0111 -> 10
1000 -> 00
1001 -> 00
1010 -> 01
1011 -> 00
1100 -> 00
1101 -> 00
1110 -> 01
1111 -> 00
Zeit: 0.0042
$nandu5.txt$ Lösung:
Start -> End
000000 -> 00010
000001 -> 00010
000010 -> 00011
000011 -> 00011
000100 -> 00100
000101 -> 00100
000110 -> 00011
000111 -> 00011
001000 -> 00010
001001 -> 00010
001010 -> 00011
001011 -> 00011
001100 -> 00100
001101 -> 00100
001110 -> 00011
001111 -> 00011
010000 -> 00010
010001 -> 00010
010010 -> 00011
010011 -> 00011
010100 -> 00100
010101 -> 00100
010110 -> 00011

010111 -> 00011
011000 -> 00010
011001 -> 00010
011010 -> 00011
011011 -> 00011
011100 -> 00100
011101 -> 00100
011110 -> 00011
011111 -> 00011
100000 -> 10010
100001 -> 10010
100010 -> 10011
100011 -> 10011
100100 -> 10100
100101 -> 10100
100110 -> 10011
100111 -> 10011
101000 -> 10010
101001 -> 10010
101010 -> 10011
101011 -> 10011
101100 -> 10100
101101 -> 10100
101110 -> 10011
101111 -> 10011
110000 -> 10010
110001 -> 10010
110010 -> 10011
110011 -> 10011
110100 -> 10100
110101 -> 10100
110110 -> 10011
110111 -> 10011
111000 -> 10010
111001 -> 10010
111010 -> 10011
111011 -> 10011
111100 -> 10100
111101 -> 10100
111110 -> 10011
111111 -> 10011
Zeit: 0.0315

2.2 Quellcode

```
1 import time
2
```

```
def read_construction_file(filename): #read the construction file and return the number of rows, number of columns, the construction, and the number of lights
    lightcount = 0 #number of lights
    with open(filename, 'r') as f: #open the file
        n, m = map(int, f.readline().strip().split()) #read the first line and get the number of rows and columns
        construction = [list(f.readline().strip().split()) for _ in range(m)] #read the construction

        for i in construction[0]: #for each item in the first row of the construction
            if i.startswith('Q'): #if the item starts with Q
                lightcount += 1 #increment the number of lights
    return n, m, construction, lightcount #return the number of rows, number of columns, the construction, and the number of lights

def interpretConstruction(n, m, construction, startingLight, lightcount): #interpret the construction and return the final state of the LEDs
    lightrow = ['O'] * n #create a list of length n with all items set to O
    for i in range(len(construction[0])): #for each item in the first row of the construction
        if construction[0][i].startswith('Q'): #if the item starts with Q
            idx = int(construction[0][i][-1]) - 1 #get the index of the light
            if 0 <= idx < lightcount: #if the index is valid
                lightrow[i] = 'l' if startingLight[idx] == '1' else 'O' #set the item at the index to 1 if the light is on or O if the light is off
        if construction[0][i].startswith('X'): #if the item starts with X
            lightrow[i] = 'O' #set the item to O
    startrow = lightrow.copy() #create a copy of the lightrow
    for i in construction[1:-1]: #for each row in the construction except the first and last
        j = 0 #set j to 0
        while j in range(len(i)): #while j is in range of the length of the row
            if j < len(i) - 1: #if j is less than the length of the row minus 1
                if i[j] == 'R' and i[j+1] == 'r': #if the item at j is R and the item at j+1 is r logic for Red Left Input
                    if lightrow[j] == 'l':
                        lightrow[j] = 'O'
                        lightrow[j+1] = 'O'
                    else:
                        lightrow[j] = 'l'
                        lightrow[j+1] = 'l'
                if i[j] == 'r' and i[j+1] == 'R': #if the item at j is r and the item at j+1 is R logic for Red Right Input
                    if lightrow[j+1] == 'l':
                        lightrow[j] = 'O'
                        lightrow[j+1] = 'O'
                    else:
                        lightrow[j] = 'l'
                        lightrow[j+1] = 'l'
            if i[j] == 'W' and i[j+1] == 'W': #if the item at j is W and the item at j+1 is W logic for White
                if lightrow[j] == 'O' and lightrow[j+1] == 'O':
```

```
                    lightrow[j] = '1'
                    lightrow[j+1] = '1'
                elif lightrow[j] == '1' and lightrow[j+1] == '1':
                    lightrow[j] = 'O'
                    lightrow[j+1] = 'O'
                else: #logic for Blue
                    lightrow[j] = '1'
                    lightrow[j+1] = '1'
                j += 1
            if i[j] == 'X': #if the item at j is X
                lightrow[j] = 'O'
            j += 1
        #print(lightrow)
    final_leds = ['1' if lightrow[led_item] == '1' and construction[-1][led_item].startswith("L") else 'O' if construction[-1][led_item].startswith("L") else '' for led_item in range(len(lightrow))] #create a list of length n with all items set to L if the light is on or X if the light is off
    return final_leds

def test_all_flashlight_combinations(n, m, construction, lightcount): #test all flashlight combinations
    print("Testing all flashlight combinations: ") #print the message
    print("Start" + " -> " + "End") #print direction
    for i in range(2**lightcount): #for each flashlight combination
        flashlights = bin(i)[2:].zfill(lightcount) #get the binary representation of the number and pad it with 0s to the length of the number of lights
        final_leds = interpretConstruction(n, m, construction, flashlights, lightcount) #interpret the construction
        print(flashlights + " -> " +  "".join(final_leds)) #print the flashlight combination and the final state of the LEDs

start = time.time() #start the timer
n, m, construction, lightcount = read_construction_file("nandu/Input/nandu5.txt") #read the construction file and get the number of rows, number of columns, the construction, and the number of lights
test_all_flashlight_combinations(n, m, construction, lightcount) #test all flashlight combinations
end = time.time() #end the timer
print('{:5.3f}s'.format(end-start), end='  ')
```