UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE BOGOTÁ
FACULTAD DE INGENIERÍA

Algoritmos
Germán J. Hernández

Taller 0
Pancakes
Miguel Angel Ortiz Marin - 1032504680 Gr 1
Edder Hernandez Forero - 1014297550 Gr 1
Edwin Mahecha Parra - 1013679888 Gr 2

1. Complete the flips and number of flips for the 4 and 5 element permutations in the files pancakes4.xls, pancakes4.xml, pancakes5.xls and pancakes5.xml (Check the examples files for 2 and 3 permutations in pancakes2.xls, and pancakes3.xls).

pancakes4.xls

Permutation	Flips		Permutation	Flips	# Flips
1234			1234		0
4321	0		1243	2	1
3214	1		1324	121	3
2134	2		1342	21	2
2341	01		1423	12	2
3421	02		1432	1	1
4123	10		2134	2	1
2314	12		2143	101	3
1432	010		2314	12	2

			2341	01	2
			2413	1021	4
			2431	210	3
			3124	1	1
			3142	1201	4
			3214	010	3
			3241	120	3
			3412	202	3
			3421	02	2
			4123	10	2
			4132	012	3
			4213	021	3
			4231	0121	4
			4312	02	2
			4321	0	1

pancakes4.xml

```
<pancakes>
<case flips="" permutation="0123"/>
<case flips="0" permutation="3210"/>
<case flips="1" permutation="0321"/>
<case flips="2" permutation="0132"/>
<case flips="01" permutation="3012"/>
<case flips="02" permutation="3201"/>
<case flips="10" permutation="1230"/>
<case flips="12" permutation="0312"/>
<case flips="20" permutation="2310"/>
<case flips="21" permutation="0231"/>
<case flips="010" permutation="2103"/>
<case flips="012" permutation="3021"/>
<case flips="020" permutation="1023"/>
<case flips="021" permutation="3102"/>
<case flips="101" permutation="1032"/>
<case flips="102" permutation="1203"/>
<case flips="120" permutation="2130"/>
```

```
<case flips="121" permutation="0213"/>
<case flips="201" permutation="2013"/>
<case flips="202" permutation="2301"/>
<case flips="210" permutation="1320"/>
<case flips="0121" permutation="3120"/>
<case flips="1021" permutation="1302"/>
<case flips="1201" permutation="2031"/>
</pancakes>
```

pancakes5.xls

Permutation	Flips		Permutation	Flips	# Flips
12345			12345		0
54321	0		12354	3	1
			13245	131	3

pancakes5.xml

```
<pancakes>
<case flips="01021" permutation="42153"/>
<case flips="01232" permutation="51324"/>
<case flips="01320" permutation="24315"/>
<case flips="02023" permutation="32514"/>
<case flips="02120" permutation="21435"/>
<case flips="02132" permutation="53142"/>
<case flips="02310" permutation="41325"/>
<case flips="02312" permutation="52413"/>
<case flips="10131" permutation="24351"/>
<case flips="10201" permutation="42315"/>
<case flips="10321" permutation="24153"/>
<case flips="12021" permutation="42513"/>
<case flips="12301" permutation="31524"/>
<case flips="13031" permutation="35142"/>
<case flips="20123" permutation="31425"/>
<case flips="20213" permutation="35241"/>
<case flips="21032" permutation="25314"/>
<case flips="21320" permutation="42531"/>
<case flips="23012" permutation="41352"/>
<case flips="32102" permutation="24135"/>
</pancakes>
```

2.Write a Python program for one spatula flips that receives the size of the permutation *n* numbers and control parameter *c*, if the parameter *c* is 0 writes *Pn*; if the parameter is *c* is 1 writes *Pn* and all the permutations (in xls and xml) that require *Pn* flips ant the corresponding flips; and if the parameter *c* is 2 writes *Pn* and all the permutations (in xls and xml) .

La función longest path del código es la que realiza la función especificada por el punto, el resto del código son funciones auxiliares para lograr esto.

El código crea exitosamente un archivo csv (para ser abierto con excel) y un archivo xml que varían su contenido de acuerdo al parámetro c de la función.

Código:

```
# -*- coding: utf-8 -*-
"""
Created on Mon Jun  4 21:27:27 2018

@author: Miguel Angel
"""
import itertools as it
from graphviz import Graph
import xml.etree.cElementTree as ET

g = Graph(format='pdf')

def flip(p, i):
    return p[0:i] + p[i:][::-1]

def flips(p):
    for i in range(len(p)-1):
        f = flip(p,i)
        yield ''.join( map(str, f) )

def flipsTwoSpatula(p):
    for i in range(len(p)-1):
        f = flip(p, i)
        for j in range (i+1, len(p) - 1):
            yield ''.join( map(str, flip(f, j) ) )

def findFlipNo(parent, son):
    i = 0
    for flip in flips(parent):
        if(flip == son):
```

```
            return i
        i+=1

def construct_network(p, spatula):
    n = len(p)
    vertices = []
    adjacents = {}
    for perm in it.permutations(p, n):
        perm = ''.join( map(str,perm) )
        g.node(perm, perm )
        adjs = []
        vertices.append(perm)
        for flip in spatula(perm):
            adjs.append(flip)
        adjacents[perm] = adjs
    return vertices, adjacents


n = 4
p = list(range(n))

flips(p)
vertices, adjacents = construct_network(p, flips)
visited = {}

def DFS(root, adjacents):
    visited[root] = True
    adjacent = adjacents[root]
    ans = 0
    for adj in adjacent:
        if not visited[adj]:
            ans = max(ans, 1 + DFS(adj, adjacents))
    return ans

def BFS(root, adjacents):
    dist = {}
    path = {}
    path[root] = ''
    dist[root] = 0
    visited[root] = True
    queue = [root]
    ans = 0
    levels = [[]for i in range(21) ]
    levels[0].append(root)
    while queue:
```

```
        u = queue.pop(0)
        adj = adjacents[u]
        for v in adj:
            if not visited[v]:
                dist[v] = 1 + dist[u]
                path[v] = path[u] + str(findFlipNo(u,v))
                g.edge(u, v)
                levels[dist[v]].append(v)
                ans = max(ans, dist[v])
                queue.append(v)
                visited[v] = True
    levels = levels[:ans+1]
    return ans, levels, path

#print(vertices[0])

class node:

    def __init__(self, data, flips):
        self.data = data
        self.flips = flips
        self.flipsno = len(flips)

    def __str__(self):
        return self.data + ',' + self.flips + ',' + str(self.flipsno)

def longest_path(vertices, adjacents, c = 0):
    for v in vertices:
        visited[v] = False
    max_len, levels, path = BFS(vertices[0], adjacents)
    #print(levels)
    #print(path)
    if(c == 0):
        print(max_len)
    elif(c == 1):
        level = levels[max_len]
        text = []
        for perm in level:
            text.append(perm + ',' + path[perm])
        with open('pancakes.csv','wb') as file:
            head = 'permutation,flips\n'
            file.write(head.encode())
            for l in text:
                file.write(l.encode())
                file.write('\n'.encode())
```

```
        pancakes = ET.Element("pancakes")
        for l in text:
            l = l.split(',')
            ET.SubElement(pancakes,'case', {'permutation':l[0],
'flips':l[1]})
        tree = ET.ElementTree(pancakes)
        tree.write("pancakes.xml")
        #print(levels[max_len])
    elif(c == 2):
        text = []
        for level in levels:
            for perm in level:
                text.append(perm + ',' + path[perm] + ',' +
str(len(path[perm])) )

        with open('pancakes.csv','wb') as file:
            head = 'permutation,flips,flipsno\n'
            file.write(head.encode())
            for l in text:
                file.write(l.encode())
                file.write('\n'.encode())
        pancakes = ET.Element("pancakes")
        for l in text:
            l = l.split(',')
            ET.SubElement(pancakes,'case', {'permutation':l[0],
'flips':l[1]})
        print('tree')
        tree = ET.ElementTree(pancakes)
        tree.write("pancakes.xml")
        #print(levels)

longest_path(vertices, adjacents,2)

g.view()
```

3. Write a Python program that does the same that is requested in problem 2 but for flips with two spatulas.

Sólo es necesario crear una función extra que simula la función de dos espátulas y hacer que la función que construye la red use esta función en vez de la otra con una sola espatula.

```
def flipsTwoSpatula(p):
    for i in range(len(p)-1):
```

```
        f = flip(p, i)
        for j in range (i+1, len(p) - 1):
            yield ''.join( map(str, flip(f, j) ) )
def construct_network(p, spatula):
    n = len(p)
    vertices = []
    adjacents = {}
    for perm in it.permutations(p, n):
        perm = ''.join( map(str,perm) )
        g.node(perm, perm )
        adjs = []
        vertices.append(perm)
        for flip in spatula(perm):
            adjs.append(flip)
        adjacents[perm] = adjs
    return vertices, adjacents
```