Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ciencias de la Computación

Tarea 1

Matemáticas Discretas para la Computación

Autor: Lukas Pavez
RUT: 19.401.577-1
Profesor: Pablo Barcelo B.
Auxiliares: Arniel Labrada D.
Ilana Mergudich T.
Ayudantes: Jaime Salas T.
Javier Marinković
Joaquín Romero
Pablo Paredes Haz
Fecha de entrega: 19/06/2018

1. P1

El lenguaje utilizado es Python 2.7, código adjunto en sección Anexos

El código desarrollado hace lo siguiente:

Primero verifica si el string entregado sigue las reglas de arit, esto es que entre cada suma o multiplicación haya al menos una variable u un número, o que no hayan dos símbolos juntos (ver función arit(s)).

Luego cuenta si el número de paréntesis izquierdos es igual al de paréntesis derechos, además, si hay uno derecho que no tiene su paréntesis izquierdo correspondiente entrega 'NO', si el numero de los paréntesis coinciden se le agrega a la respuesta una 'c' por cada par de paréntesis (ver función countBrackets(s)).

Después agrega una a por cada constante (countConsts(s)), una d por cada símbolo -", una e por cada "*ƨ una b por cada letra.

Finalmente ordena alfabéticamente la respuesta obtenida y la entrega.

Para probar el código, al correr el programa, se pregunta al usuario si quiere probar ingresando solo un string o probar con un archivo que contiene inputs y otro con los respectivos outputs, para esto el usuario debe ingresar 0 o 1 respectivamente.

La primera opción que hace un print de p1(string), donde p1 es la función que entrega la respuesta.

La segunda opción llama a la función test(In, Out), que recibe ambos archivos, por cada linea de ambos archivos llama a p1(linea_input), y luego lo compara con la linea del archivo de output ordenada alfabéticamente, esta función hace un print con el numero de errores que hubo, junto con los strings que causaron el error.

2. P2

El lenguaje utilizado es Python 2.7, código adjunto en sección Anexos

2.1. a

El código desarrollado hace lo siguiente:

Primero se pasa el string a una matriz con listas de python, como piden ver si la relación es un cuasi-orden, hay que verificar que es refleja y transitiva, para verificar que es refleja se recorre la diagonal de la matriz, y si hay un 0 entonces ya no es refleja, para verificar transitividad se hacen 3 ciclos for para generar las combinaciones necesarias para verificar que se cumpla la condición de transitividad.

2.2. b

R es un cuasi-orden, por lo tante es refleja y transitiva.

Refleja:
Como R es refleja,para todo a $\in$ A, (a,a) $\in$ R, por esto también se tiene que (a,a) $\in R^{-1}$, por lo tanto, $R \cap R^{-1}$ es refleja.

Simétrica:
Para un par (a,b) $\in R \cap R^{-1}$, se tiene el par (b,a) $\in R \cap R^{-1}$, ya que es la intersección de una relación con su inverso, por lo tanto $R \cap R^{-1}$ es simétrica.

Transitividad:
Si se toma un par (a,b) $\in R \cap R^{-1}$ y un par (b,c) $\in R \cap R^{-1}$, se tiene que por transitividad (a,c) $\in R$, y como $R \cap R^{-1}$ es simétrico, también se tienen los pares (c,b) y (b,a) $\in R \cap R^{-1}$, que también están en R, por lo que se tiene que (a,c) $\in R$ y (c,a) $\in R$, por lo tanto (c,a) $\in R \cap R^{-1}$, entonces $R \cap R^{-1}$ es una relación transitiva.

Finalmente, $R \cap R^{-1}$ es la mayor relación de equivalencia contenida en R, ya que en el caso de que R no es simétrica, $R \cap R^{-1}$ está contenida en R y es una relación de equivalencia, y si es simétrica, $R = R \cap R^{-1}$, y como también es un cuasi-orden, R sería una relación de equivalencia.

2.3. c

Lo que hace el código es traspasar el string recibido a una matriz con listas de python, hacer la intersección con la matriz inversa y luego entregar el resultado en formato string.

3. P3

El lenguaje utilizado es Python 2.7, código adjunto en sección Anexos

Para la función sat_dnf(F), se hizo una lista donde cada elemento es una clausula de la oración entregada, luego se revisa si en una de estas clausulas se encuentran una variable y su negación, si esto ocurre, es insatisfacible, si no, es satisfacible.

Para la función sat_2cnf se separó la oración en por clausula, luego cada una de estas clausulas se transformaron a implicancias ((x1ox2) pasa a ser $(-x1 \Rightarrow x2) \wedge (-x2 \Rightarrow x1)$), donde los vertices del grafo son el lado izquierdo de la implicancia, luego se construyo un grafo dirigido con las implicancias, para después por cada vértice del grafo se busca si es que existe un camino desde el vértice a su negación y un camino desde su negación al vértice, si esto pasa el problema es insatisfacible, ya que se tendría $(x \Rightarrow -x) \wedge (-x \Rightarrow x)$, y esto es una contradicción.

4. P4

El lenguaje utilizado es Python 2.7, código adjunto en sección Anexos

4.1. a

Se realizará la demostración utilizando inducción:

Caso base $n = 2$:
Para n=2, se tiene un tablero de 4x4, donde la máxima cantidad de caballos que se pueden poner sin que se coman es poniéndolos intercalados, así quedarían 4 caballos por fila, por 4 filas serían 8 caballos, que es igual a 2^{2*2-1}, por lo tanto se cumple el caso base.

Ahora se supone que se cumple la hipótesis inductiva para n, esto es, en un tablero de $2^n x 2^n$ se pueden poner a lo más 2^{2n-1} caballos.

Ahora se demostrará que se cumple para n+1:
Si se tiene un tablero de $2^{(n+1)}x2^{(n+1)}$, se puede dividir en 4 partes iguales, por lo que se forman 4 tableros de tamaño $2^n x 2^n$, y por hipótesis inductiva la cantidad máxima de caballos que se pueden poner en cada uno de estos tableros es de 2^{2n-1}, y al sumar los caballos de estos cuatro tableros se tiene como máximo $4 * 2^{2n-1} = 2^{2n+1} = 2^{2(n+1)-1}$ caballos.

Por lo tanto en un tablero de tamaño $2^{(n+1)}x2^{(n+1)}$ se pueden poner a lo más $2^{2(n+1)-1}$ caballos sin que se coman entre ellos.

4.2. b

El código realizado crea una matriz de tamaño $2^n x 2^n$ rellena de 0s, y se ponen $2^n - 1$ 1s intercalados en cada fila lo que dejaría $2^n x 2^n - 1 = 2^2 n - 1$ caballos.

5. Anexos

5.1. P1

```python
#!/usr/bin/env python

def p1(s):
    letters = "qwertyuiopasdfghjklzxcvbnm"
    const = "a"
    var   = "b"
    par   = "c"
    su    = "d"
    mult  = "e"

    res = ""

    if not arit(s):
        return "NO\n"

    pars = countBrackets(s)
    if pars == "NO\n":
        return pars

    res += par*pars # agrega parentesis
    res += const*countConsts(s) # agrega constantes
    res += su*s.count("+") # agrega sumas
    res += mult*s.count("*") # agrega multiplicaciones

    for a in s: # agrega variables, suponiendo que cada variable esta compuesta por una letra
        if a in letters:
            res += var

    res = ''.join(sorted(res)) # ordena respuesta final alfabeticamente
    return res

def arit(s):
    letters = "qwertyuiopasdfghjklzxcvbnm"
    num = "1234567890"
    l = s.split("+")
    if '' in l:
        return False
    b = True
    for st in l:
        c = False
        if st[0] == '*' or st[len(st) - 1] == '*': return False
        for a in st:
```

```
            if a in letters or a in num:
                c = True
        b = b and c

    l = s.split("*")
    if '' in l:
        return False
    d = True
    for st in l:
        c = False
        if st[0] == '+' or st[len(st) - 1] == '+': return False
        for a in st:
            if a in letters or a in num:
                c = True
        d = d and c
    return b and d

def countConsts(s):
    num = "1234567890"

    n = 0
    i = 0
    l = len(s)
    while i<l:
        if s[i] in num:
            n += 1
            try:
                while s[i+1] in num:
                    i += 1
            except Exception as e:
                pass
        i += 1
    return n

def countBrackets(s):
    pars = 0
    for a in s:
        if a == "(":
            pars += 1
        if a == ")":
            pars -= 1
        if pars < 0:
            return "NO\n"
    if pars == 0:
        return s.count("(")
    else:
        return "NO\n"
```

```
def test(In, Out):
    file_in  = open(In,"r")
    file_out = open(Out,"r")

    f = 0
    for line in file_out:
        line_in = file_in.readline()

        if line == "NO\n":
            res = "NO\n"
        else:
            # ordena alfabeticamente
            res = line.replace(",","").replace("\n","")
            res = ''.join(sorted(res))

        bol = p1(line_in) == res
        if not bol:
            print line_in
            f+=1
    print "Errores: " + str(f)




if __name__ == '__main__':
    print "-------------------------------------------------------------------------"
    b = input("probar con un string / probar con archivos con input-outputs [0/1]: ")
    print "-------------------------------------------------------------------------"
    if b==0:
        #s = "(90*r+b+(l*y)*95)*j*b*((g*51)+x)+29+g"
        s = raw_input("ingrese string: ")
        print "-------------------------------------------------------------------------"
        print p1(s)
    else:
        In = raw_input("ingrese nombre de archivo con inputs (ej: Arit_In.txt): ")
        Out = raw_input("ingrese nombre de archivo con outputs (ej: Arit_Out.txt): ")
        print "-------------------------------------------------------------------------"
        test(In, Out)

    print "-------------------------------------------------------------------------"
```

5.2. P2

```python
#!/usr/bin/env python


###################### PARTE a ######################

def p2_a(s):
    #cuasi orden: refleja y transitiva
    R = stringToMatrix(s)

    if refleja(R) and transitiva(R):
        return 1
    return 0

def refleja(R):
    #refleja: R(i, i) = 1
    res = True
    n = len(R)
    #recorre matriz triangular superior
    for i in range(0, n):
        if R[i][i] == '0':
            res = False
            break
    return res

def transitiva(R):
    #transitiva: if R(i, j) and R(j, k) => R(i, k)
    res = True
    n = len(R)
    for i in range(0, n):
        for j in range(0, n):
            for k in range(0, n):
                if (R[i][j] == '1') and (R[j][k] == '1') :
                    if R[i][k] != '1' :
                        res = False
                        break;
    return res



###################### PARTE c ######################


def p2_c(s):
    R = stringToMatrix(s)
    return matrixToString(RinterRinv(R))
```

```
def RinterRinv(R):
    l = len(R)
    res = []
    Rinv = inv(R)

    #inicializar matriz con 0s
    for i in range(0, l):
        res.append(['0' for i in range(0, l)])

    #hacer interseccion
    for i in range(0, l):
        for j in range(0, l):
            if R[i][j] == Rinv[i][j] == '1':
                res[i][j] = '1'
    return res


def inv(R):
    l = len(R)
    res = []
    #inicializar matriz con 0s
    for i in range(0, l):
        res.append(['0' for i in range(0, l)])

    #construir Rinv
    for i in range(0, l):
        for j in range(0, l):
            if R[i][j] == '1':
                res[j][i] = '1'
    return res


###################### MISC ######################

def stringToMatrix(s):
    #split no funciona bien con \n cuando se usa raw_input ???
    l = s.split('\\n')
    m = []
    for i in range(0, len(l)):
        m.append(l[i].split(" "))
    return m

def matrixToString(M):
    res = ""
    l = len(M)
    for i in range(0, l):
        for j in range(0, l):
```

```
            res += M[i][j] + " "
        res += '\n'
    return res


if __name__ == '__main__':
    print "-------------------------------------------------------"
    print "--------------------- p2_a ---------------------\n"
    s = raw_input("matriz: ")

    print '\nrespuesta: ' + str(p2_a(s))

    print "\n-------------------------------------------------------"
    print "--------------------- p2_c ---------------------\n"

    #1 1 1\n0 1 1\n0 1 1
    s = raw_input("matriz: ")
    print '\nrespuesta:\n' + p2_c(s)

    print "-------------------------------------------------------"
```

5.3. P3

5.3.1. SAT

```
#!/usr/bin/env python

from Graph import *

def sat_2cnf(F):
    graph = Graph()

    fill_graph(graph, F)

    keys = graph.get_vertexs()
    for key in keys:
        p1 = graph.find_path(key, neg(key))
        p2 = graph.find_path(neg(key), key)
        if p1 != None and p2 != None:
            return 0
    return 1

def sat_dnf(F):
    l = F.replace('(', '').replace(')', '').split('o')
    for c in l:
        n = c.split('y')
```

```
        for v in n:
            if neg(v) in n:
                return 0
    return 1

def fill_graph(graph, F):
    l = F.replace('(', '').replace(')', '').split('y')

    v = []
    for c in l:
        n = c.split('o')
        A = n[0]
        B = n[1]
        add_v(neg(A), v, graph)
        add_v(neg(B), v, graph)
        graph.add_edge(neg(A), B)
        graph.add_edge(neg(B), A)

def add_v(ver, v, graph):
    if not ver in v:
        graph.add_vertex(ver)
        v.append(ver)

def neg(v):
    if v[0] == '-':
        return v[1:]
    else:
        return '-' + v


if __name__ == '__main__':
    print "-----------------------------------------------------"
    print "---------------------- test cnf ----------------------\n"

    print sat_2cnf("(x1ox2)y(-x2ox3)y(-x1o-x2)y(x3ox4)y(-x3ox5)y(-x4o-x5)y(-x3ox4)")
    #print sat_2cnf("(-x1ox5)y(-x5o-x1)y(x2ox4)y(x1ox3)y(-x3ox1)")

    print "\n-----------------------------------------------------"
    print "---------------------- test dnf ----------------------\n"

    print sat_dnf("(x1y-x2)o(x2y-x1yx4)o(-x4y-x3yx1)")

    print "\n-----------------------------------------------------"
```

5.3.2. Graph

```
#!/usr/bin/env pythonZ
#Graph implementation obtained from http://www.forosdelweb.com/f130/aporte-sencilla-implementacion-grafos-817941/

class Graph:
    #Simple graph implementation:
    #Directed graph
    #Without weight in the edges
    #Edges can be repeated

    def __init__(self):
        self.graph = {}

    def add_vertex(self, vertex):
        #Add a vertex in the graph
        #Overwrite the value
        self.graph[vertex] = []

    def get_vertexs(self):
        #Get the vertexs in the graph
        return self.graph.keys()

    def del_vertex(self, vertex):
        #Remove the vertex if it's in the graph
        try:
            self.graph.pop(vertex)
        except KeyError:
            #Here vertex is not in graph
            pass
    def is_vertex(self, vertex):
        #Return True if vertex is in the graph
        #otherwise return False
        try:
            self.graph[vertex]
            return True
        except KeyError:
            return False

    def add_edge(self, vertex, edge):
        #Add a edge in vertex if vertex exists
        try:
            self.graph[vertex].append(edge)
        except KeyError:
            #Here vertex is no in graph
            pass
```

```
    def delete_edge(self, vertex, edge):
        #Remove a edge in vertex
        try:
            self.graph[vertex].remove(edge)
        except KeyError:
            #Here vertex is not in graph
            pass
        except ValueError:
            #Here the edge not exists
            pass

    def get_edge(self, vertex):
        #Return the edges of a vertex if the vertex is in the graph
        #Otherwise return None
        try:
            return self.graph[vertex]
        except KeyError:
            pass

    def __str__(self):
        #Print the vertex
        s = "Vertex -> Edges\n"
        for k, v in self.graph.iteritems():
            s+= "%-6s -> %s\n" % (k, v)
        return s


###### implementation obtained from https://www.python.org/doc/essays/graphs/

    def find_path(self, start, end, path=[]):
        path = path + [start]
        if start == end:
            return path
        if not self.graph.has_key(start):
            return None
        for node in self.graph[start]:
            if node not in path:
                newpath = self.find_path(node, end, path)
                if newpath: return newpath
        return None
```

5.4. P4

```
#!/usr/bin/env python

import numpy as np
```

```
def p4_b(n):
    l = 2**n
    res = np.zeros((l, l))
    b = True
    for i in range(0, l):
        for j in range(0, l):
            if b:
                res[i][j] = 1
            b = not b
        b = not b
    return toString(res)

def toString(m):
    res = ""
    l = np.shape(m)[0]
    for i in range(0, l):
        for j in range(0, l):
            if m[i][j] == 1:
                res += '1'
            else:
                res += '0'
            res += ' '
        res += '\n'
    return res

if __name__ == '__main__':
    print "-------------------------------------------------"

    n = input('n: ' )
    print p4_b(n)

    print "-------------------------------------------------"
```