Heuristica

June 16, 2018

In [1]:
```
import pyspark
import copy
from IPython.display import display
import time
```

1 Declaracion de funciones

In [2]:
```
#-----------------------------------------------------------------
# GET RELATION ACTIVITIES CATEGORIES
#-----------------------------------------------------------------
def getRelationActivitiesCategories(tuplesList, activitiesDict):
    """
    Pre: Recibo un diccionario de actividades,
        donde la clave es el nombre de la actividad, y
        su valor  es su numero de hash. Lo mismo para
        el dicionario de categorias.
     Pos: Recibo un diccionario relacion donde la clave es la
         cagtegoria y su valor es una lista de las actividades
         que perteneces a dicha categoria.
     """
    size = len(tuplesList)
    relation = {}
    for i in range(0, size):
        activity = tuplesList[i][0]
        category = tuplesList[i][1]
        if category not in relation:
            relation[category] = [activitiesDict[activity]]
        else:
            relation[category].append(activitiesDict[activity])
    return relation
#-----------------------------------------------------------------
# GET HASH NAMES
#-----------------------------------------------------------------
def gethHashedNames(namesList):
    """
    pre: Obtengo una lista de nombres (tokens)
    pos: Retorno un dicionarioa que por cada token como clave
```

```
        le asigna una clave que sera su hash
    """
    names = {}
    cont = 0
    size = len(namesList)
    for i in range(0, size):
        if namesList[i] in names: continue
        names[namesList[i]] = cont
        cont += 1
    return names
#-----------------------------------------------------------------
# HASH NAMES
#-----------------------------------------------------------------
def hashNames(x, pos):
    """
    pre: Obtengo una lista de listas (x) que representa todos los
        registros del rdd y la posicion en que quiero hashear
        de cada registro de la lista.
    pos: Retorno la lista de lista modificada con el hash
        correspondiente
    """
    names = {}
    cont = 0
    x = list(x)
    size = len(x)
    for i in range(0, size):
        if x[i][pos] in names:
            x[i][pos] = names[x[i][pos]]
            continue
        names[x[i][pos]] = cont
        cont += 1
        x[i][pos] = names[x[i][pos]]
    return x
#-----------------------------------------------------------------
# TOOK TIMES TO SECONDS
#-----------------------------------------------------------------
def TookTimesToSeconds(aTookTime):
    """
    pre: Obtengo un took time con este
        formato '2018-04-21T16:22:49.000Z'
    pos: Retorno el tooktime en segundos
    """
    # '2018-04-21T16:22:49.000Z'
    aTookTime = aTookTime.split(aTookTime[10])[1]
    aTookTime = aTookTime.split(aTookTime[8])[0]
    #16:22:49
    aTookTime = aTookTime.split(':')
    aTookTime[0] = int(aTookTime[0])
```

```
    aTookTime[1] = int(aTookTime[1])
    aTookTime[2] = int(aTookTime[2])
    if (aTookTime[0] < 13):
        aTookTime[0] += 24
    aTookTime[0] = aTookTime[0]*60*60
    aTookTime[1] = aTookTime[1]*60
    aTookTime = aTookTime[0] + aTookTime[1] + aTookTime[2]
    return aTookTime
#-------------------------------------------------------------------
# CHANGE TOOK TIMES
#-------------------------------------------------------------------
def changeTookTimes(x):
    """
    pre: Obtengo todos los registros de un rdd
    pos: Retorno los registos modificdos en los tooktimes donde
        cada uno esta en segundos y entre tooktimes esta la diferencia
        o distancia de tiempo en segundos entre uno y otro
    """
    size = len(x[1])
    # name, [(w, t), (w, t), ....]
    for j in range(0, size): x[1][j][1] = TookTimesToSeconds(x[1][j][1])
    vec = []
    for j in range(0, size):
        if (j == 0): vec.append([x[1][j][0], 0])
        else:
            newTookTime = x[1][j][1] - x[1][0][1]
            # 14:00:00: 840 -> 0 -> vec[0]
            # 14:01:00: 900 -> 60 ->
            # 14:02:00: 960 -> 120
            # 14:05:00:  -> 300
            vec.append([x[1][j][0], newTookTime])
    x[1] = vec
    return x
#-------------------------------------------------------------------
# CHANGE FORMAT
#-------------------------------------------------------------------
def changeFormat(x):
    size = len(x[1])
    for j in range(0, size):
        x[1][j] = [x[0], x[1][j][0], x[1][j][1]]
    return x[1]
#-------------------------------------------------------------------
# DELETE REPETITIONS
#-------------------------------------------------------------------
def deleteRepetitions(x):
    repeated = {}
    indexToDelete = []
    size = len(x[1])
```

```
    for i in range(0, size):
        tookTime = x[1][i][1]
        if tookTime not in repeated:
            repeated[tookTime] = True
        else:
            indexToDelete.append(i)
    for i in indexToDelete:
        x[1].pop(i)
    return x
#---------------------------------------------------------------------
# SHOW ERRORS
#---------------------------------------------------------------------
def showErrors(disney):
    disney = disney.map(lambda x: (x[2], (1, x[0])))
    disney = disney.reduceByKey(lambda x, y: (x[0]+y[0], x[1]))
    disney = disney.collect()
    return [x for x in disney if x[1][0] != 22]
#---------------------------------------------------------------------
# REMOVE TOOK TIMES
#---------------------------------------------------------------------
def removeTookTimes(disney, listTookTimes):
    return disney.filter(lambda x: x[2] not in listTookTimes)
#---------------------------------------------------------------------
# GET TOOK TIMES MINUTES
#---------------------------------------------------------------------
def getTookTimeMinutes(aTookTime):
    #'2018-04-21T13:14:57.000Z'
    aTookTime = aTookTime.split(aTookTime[10])[1]
    aTookTime = aTookTime.split(aTookTime[8])[0]
    #13:14:57
    aTookTime = aTookTime.split(':')
    hours = int(aTookTime[0])
    if (hours < 13):
        hours += 24
    minutes = int(aTookTime[1])
    return hours*60 + minutes
#---------------------------------------------------------------------
# REMOVE TOOK TIMES EVERY GAP
#---------------------------------------------------------------------
def removeTookTimesEveryGap(reg, gap):
    vec = []
    size = len(reg[1])
    vec.append(reg[1][0])
    T0 = getTookTimeMinutes(reg[1][0][1])
    for i in range(1, size):
        T1 = getTookTimeMinutes(reg[1][i][1])
        if (T1 - T0 >= gap):
            T0 = T1
```

```
            vec.append(reg[1][i])
    reg[1] = vec
    return reg
#---------------------------------------------------------------------
# SHOW TOOK TIME HASH
#---------------------------------------------------------------------
def showTookTimehash(tookTimes, init, end):
    size = len(tookTimes)
    for i in range(init, end):
        print i, ": ", tookTimes[i][1]
#---------------------------------------------------------------------
# GET DICT TOOK TIMES
#---------------------------------------------------------------------
def getDictTookTimes(tookTimes, disney, MAX_TOOKTIME):
    tookTimesInSeconds = sc.parallelize(disney).map(lambda x: x[2])
    tookTimesInSeconds = tookTimesInSeconds.collect()[0:MAX_TOOKTIME]
    dicc = {}
    for i in range(0, MAX_TOOKTIME):
        aTookTime = tookTimes[i][1]
        aTookTime = aTookTime.split(aTookTime[10])[1]
        aTookTime = aTookTime.split(aTookTime[8])[0]
        dicc[tookTimesInSeconds[i]] = aTookTime
    return dicc
#---------------------------------------------------------------------
# PARSE DISNEY
#---------------------------------------------------------------------
def parseDisney(gap):
    disneyRoot = "SetDeDatos/disney.csv"
    disney = spark.read.load(disneyRoot, format="csv", header=True).rdd
    disney = disney.map(lambda line: list([str(x) for x in line]))
    disney = disney.map(lambda x: [x[1], int(x[2]), x[3]])
    disney = disney.map(lambda x: [x[0], [x[1], x[2]]]).groupByKey()
    disney = disney.map(lambda x: [x[0], list(x[1])])
    disney = disney.map(lambda x: removeTookTimesEveryGap(x, gap))
    tookTimes = disney.collect()[0][1]
    disney = disney.map(lambda x: deleteRepetitions(x))
    disney = disney.map(lambda x: changeTookTimes(x))
    disney = disney.mapPartitions(lambda x: hashNames(x, 0))
    disney = disney.flatMap(lambda x: changeFormat(x))
    disney = disney.map(lambda x: [x[0], x[1]*60, x[2]])
    print "Errors: ", showErrors(disney)
    print "Tooktimes: "
    showTookTimehash(tookTimes, 0, 10)
    print "..."
    print "..."
    size = len(tookTimes)
    showTookTimehash(tookTimes, size-5, size)
    print ""
```

```
    print "Parsed Disney take(10): "
    display(disney.take(10))
    print "..."
    disney = disney.collect()
    tookTimeDicc = getDictTookTimes(tookTimes, disney, len(tookTimes))
    return disney, tookTimeDicc, tookTimes
#----------------------------------------------------------------
# PARSE DESCRIPTION
#----------------------------------------------------------------
def parseDescription():
    descriptionRoot = "SetDeDatos/descripcion.csv"
    description = spark.read.load(descriptionRoot, format="csv", header=True).rdd
    description = description.map(lambda line: list([str(x) for x in line]))
    #-------------------------------------------------------------------------------
    # Hasheo las categorias y obtengo un diccionario
    categoriesDict = gethHashedNames(description.map(lambda x: x[1]).collect())
    # Hasheo las actividades y obtengo un diccionario
    activitiesDict = gethHashedNames(description.map(lambda x: x[0]).collect())
    tupleList = description.map(lambda x: (x[0], x[1])).collect()
    relation = getRelationActivitiesCategories(tupleList, activitiesDict)
    #-------------------------------------------------------------------------------
    categoriesDict["ACCION"] = categoriesDict.pop("accion")
    categoriesDict["BOTES"] = categoriesDict.pop("botes")
    categoriesDict["FUTURO"] = categoriesDict.pop("futuro")
    categoriesDict["MONTANA"] = categoriesDict.pop("monta??a")
    categoriesDict["MUSICAL"] = categoriesDict.pop("musical")
    categoriesDict["NINOS"] = categoriesDict.pop("ni??os")

    relation["ACCION"] = relation.pop("accion")
    relation["BOTES"] = relation.pop("botes")
    relation["FUTURO"] = relation.pop("futuro")
    relation["MONTANA"] = relation.pop("monta??a")
    relation["MUSICAL"] = relation.pop("musical")
    relation["NINOS"] = relation.pop("ni??os")
    #-------------------------------------------------------------------------------
    print "Categorias: "
    display(categoriesDict)
    print " "
    print "Actividades: "
    display(activitiesDict)
    print " "
    #-------------------------------------------------------------------------------
    description = description.mapPartitions(lambda x: hashNames(x, 0))
    description = description.mapPartitions(lambda x: hashNames(x, 1))
    description = description.map(lambda x: [x[0], x[1], int(x[2]), int(x[3])])
    print "Description: "
    display(description.collect())
    #-------------------------------------------------------------------------------
```

```
    return description, activitiesDict, categoriesDict, relation
#-------------------------------------------------------------------
# COPY LIST
#-------------------------------------------------------------------
def copyList(aList, size):
    vec = copy.copy(aList)
    for i in range(0, size):
        vec[i] = copy.copy(aList[i])
    return vec
#-------------------------------------------------------------------
# DISNEY TO DICT
#-------------------------------------------------------------------
def DisneyToDict(disney):
    # disney[i] = [activity, waittime, tooktime]
    disney = sc.parallelize(disney)
    disney = disney.map(lambda x: [x[0], [x[1], x[2]]])
    disney = disney.groupByKey().map(lambda x: [x[0], list(x[1])])
    disney = disney.collect()
    dicc = {}
    for row in disney:
        dicc[row[0]] = row[1:len(row)][0]
    return dicc
#-------------------------------------------------------------------
# ADD FOOD
#-------------------------------------------------------------------
def addFood(args):
    tookTimes, activitiesDict, categoriesDict, disney, description = args
    waitTimeFood = 40*60
    SizeTookTimes = len(tookTimes)
    size = len(activitiesDict.keys())
    activitiesDict["food"] = size
    categoriesDict["food"] = len(categoriesDict.keys())
    vec = copyList(disney[0:SizeTookTimes], SizeTookTimes)
    for i in range(0, SizeTookTimes):
        vec[i][0] = activitiesDict["food"]
        vec[i][1] = waitTimeFood
    disney = disney+vec
    description = description.collect()
    description.append([activitiesDict["food"], categoriesDict["food"], 0, 1])
    description = sc.parallelize(description)
    disneyDict = DisneyToDict(disney)
    return activitiesDict, categoriesDict, disney, description, disneyDict
```

2 Heuristica

In [3]:
```
#-------------------------------------------------------------------
# THERE IS OVERLAP
```

```
#-----------------------------------------------------------------
def thereIsOverlap(tookTime, madeActivities):
    # tookTime = [waittime, tooktime]
    #madeActivities[i] = [
        #activity,
        #waittime,
        #tooktime,
        #category,
        #points,
        #imprescindible
    #]
    size = len(madeActivities)
    for i in range(0, size):
        I_a = tookTime[1]
        W_a = tookTime[0]
        F_a = I_a + W_a
        I_b = madeActivities[i][2]
        W_b = madeActivities[i][1]
        F_b = I_b + W_b
        if F_a > I_b and I_a <= I_b:
            return True
        if F_b > I_a and I_b <= I_a:
            return True
    return False
#-----------------------------------------------------------------
# IS EMPTY
#-----------------------------------------------------------------
def isEmpty(disneyDict):
    for key in disneyDict.keys():
        if (disneyDict[key]):
            return False
    return True
#-----------------------------------------------------------------
# GET KEY FROM VALUE
#-----------------------------------------------------------------
def getKeyfromValue(dicc, aValue):
    for key, value in dicc.iteritems():
        if value == aValue:
            return key
    return None
#-----------------------------------------------------------------
# COPY DICC
#-----------------------------------------------------------------
def copyDicc(dicc):
    diccCopy = {}
    for key in dicc.keys():
        diccCopy[key] = copy.copy(dicc[key])
    return diccCopy
```

```
#-----------------------------------------------------------------
# SORT DICT BY WAITTIME
#-----------------------------------------------------------------
def sortDictByWaittime(disneyDict):
    for key in disneyDict.keys():
        disneyDict[key] = sorted(disneyDict[key], key=lambda w: w[0])
    return disneyDict
#-----------------------------------------------------------------
# GET FINAL SCORE
#-----------------------------------------------------------------
def getFinalScore(madeActivities):
    size = len(madeActivities)
    finalPoints = 0
    #madeActivities[i] = [
        #activity,
        #waittime,
        #tooktime,
        #category,
        #points,
        #imprescindible
    #]
    for i in range(0, size):
        finalPoints += madeActivities[i][4]
    return finalPoints
#-----------------------------------------------------------------
# GET QUANTITY RESUME
#-----------------------------------------------------------------
def getQuantityResume(activitiesQuant, activitiesDict):
    act = []
    for HashedActivity in activitiesQuant.keys():
        activity = getKeyfromValue(activitiesDict, HashedActivity)
        act.append([activity, activitiesQuant[HashedActivity]])
    return act
#-----------------------------------------------------------------
# GET RESUME
#-----------------------------------------------------------------
def getResume(madeActivities, tookTimeDicc):
    size = len(madeActivities)
    resume = []
    #madeActivities[i] = [
        #activity,
        #waittime,
        #tooktime,
        #category,
        #points,
        #imprescindible
    #]
    for i in range(0, size):
```

```
        tookTime = madeActivities[i][2]
        madeActivities[i][2] = tookTimeDicc[tookTime]
        madeActivities[i][1] = madeActivities[i][1]/60
        activity = madeActivities[i][0]
        madeActivities[i][0] = getKeyfromValue(activitiesDict, activity)
        a = madeActivities[i][0]
        t = madeActivities[i][2]
        w = madeActivities[i][1]
        resume.append([a, t, w])
    resume.reverse()
    return resume
#---------------------------------------------------------------------
# HEURISTICA
#---------------------------------------------------------------------
def heuristica(args):

    # unpacking Arguments
    activitiesDict = args[0]
    categoriesDict = args[1]
    description = args[2]
    disneyDict = args[3]
    MAX_ACTIVITIES = args[4]
    MAX_CATEGORIES = args[5]
    MAX_MONTANA = args[6]
    MAX_NINOS = args[7]
    MAX_IMPRESCINDIBLES = args[8]
    stop = args[9]

    start = time.time()
    imprescindibles = 0
    activitiesQuant = {}
    categoriesQuant = {}
    madeActivities = []
    for i in range(0, MAX_ACTIVITIES):
        activitiesQuant[activitiesDict.values()[i]] = 0
    for i in range(0, MAX_CATEGORIES):
        categoriesQuant[categoriesDict.values()[i]] = 0

    montana = categoriesQuant[categoriesDict['MONTANA']]
    ninos = categoriesQuant[categoriesDict['NINOS']]
    MPM = 0 # vale 1 si hoce 'Mickeys PhilharMagic'
    MPMHash = activitiesDict["Mickeys PhilharMagic"]

    activitiesList = copy.copy(description)
    activityPosition = 0
    disneyDictCopy = copyDicc(disneyDict)
    # Ordeno waittimes de menor a mayor
    disneyDictCopy = sortDictByWaittime(disneyDictCopy)
```

```
while (imprescindibles < MAX_IMPRESCINDIBLES or not isEmpty(disneyDictCopy)\
       or montana < 3):
    if activityPosition >= len(activitiesList):
        activityPosition = 0
    # activitiesList[i] = [activity, category, points, imprescindible]
    activity = activitiesList[activityPosition][0]
    activityName = getKeyfromValue(activitiesDict, activity)
    category = activitiesList[activityPosition][1]
    categoryName = getKeyfromValue(categoriesDict, category)
    points = activitiesList[activityPosition][2]
    isImprescindible = activitiesList[activityPosition][3]
    activityInfo = disneyDictCopy[activity]
    i = len(activityInfo)
    while activityInfo:
        if i >= len(activityInfo):
            i = 0
        if (activityName == "food" and activitiesQuant[activity] == 1):
            activityInfo.pop(i)
            break

        tookTime = activityInfo[i]  # tookTime = [waittime, tooktime]

        if (imprescindibles < MAX_IMPRESCINDIBLES and not isImprescindible):
            break
        if not thereIsOverlap(tookTime, madeActivities):
            if (activityName == "Mickeys PhilharMagic" \
                and activitiesQuant[activity] == 1):
                MPM = 1
            if (categoryName == 'MONTANA' and \
                categoriesQuant[category] == (MAX_MONTANA - 1) and MPM == 0):
                MPMLeft = len(disneyDictCopy[MPMHash])
                if (MPMLeft == 0):
                    activityInfo.pop(i)
                break
            if (categoryName == 'NINOS' and \
                categoriesQuant[category] == (MAX_NINOS - 1)):
                activityInfo.pop(i)
                break
            activityInfo.pop(i)
            task =[activity] + tookTime + [category, points, isImprescindible]
            madeActivities.append(task)
            activitiesQuant[activity] += 1
            categoriesQuant[category] += 1
            montana = categoriesQuant[categoriesDict['MONTANA']]
            if isImprescindible and activitiesQuant[activity] == 1:
                imprescindibles += 1
            break
```

```
                else:
                    activityInfo.pop(i)
                i += 1
            if (imprescindibles == MAX_IMPRESCINDIBLES and stop):
                stop = False
                activitiesList = sorted(activitiesList , key=lambda act: -act[2])

                activityPosition = 0
            else:
                activityPosition += 1

        end = time.time()
        totalTime = end - start
        print "Time used: " + str(totalTime) + " seconds"

        ninos = categoriesQuant[categoriesDict['NINOS']]
        montana = categoriesQuant[categoriesDict['MONTANA']]
        if (imprescindibles < MAX_IMPRESCINDIBLES or montana < 3 or
            ninos >= MAX_NINOS or (MPM == 0 and montana == MAX_MONTANA)):
            print "Solucion imcompatible"

        madeActivities = sorted(madeActivities, key=lambda x: x[2])
        print "Cantidad de imprescindibles", imprescindibles, \
        "de un total de ", MAX_IMPRESCINDIBLES
        print montana, "de ", MAX_MONTANA, "montañas"
        print ninos, "de ", MAX_NINOS, "niños"
        print "Mickeys PhilharMagic: ", activitiesQuant[MPMHash]

        return madeActivities, categoriesQuant, activitiesQuant
```

3 Flujo pricipal (main)

In [4]:
```
gap = 1
carpet = "modelo" + str(gap)
dataName = carpet + "/datos" + str(gap) + ".txt"
modelName = carpet + "/model" + str(gap) + ".txt"

# Parseo csv de disney y descricion
disney, tookTimeDicc, tookTimes = parseDisney(gap)
description, activitiesDict, categoriesDict, relation = parseDescription()

# Agrego comida
args = [tookTimes, activitiesDict, categoriesDict, disney, description]
returnValues = addFood(args)
activitiesDict, categoriesDict, disney, description, disneyDict = returnValues

# Declaro constantes
```

```
        MAX_ACTIVITIES = len(activitiesDict.values())
        MAX_CATEGORIES = len(categoriesDict.values())
        MAX_IMPRESCINDIBLES = len(description.map(lambda x: [x[3], 1])\
                                  .filter(lambda x: x[0] == 1).collect())
        MAX_TOOKTIME = len(tookTimes)
        MAX_MONTANA = len(relation['MONTANA'])
        MAX_NINOS = len(relation['NINOS'])

        # Ordeno la lista de actividades de mayor a menor primero segun
        #si son imprescindibles o no, y depsues segun el puntaje
        # description[i] = [activity, category, points, imprescindible]
        description = description.collect()    # convierto rdd a lista
        description = sorted(description, key=lambda x: -x[2])
        description = sorted(description, key=lambda x: -x[3])

Errors:  []
Tooktimes:
0 :  2018-04-21T13:14:57.000Z
1 :  2018-04-21T13:15:52.000Z
2 :  2018-04-21T13:16:52.000Z
3 :  2018-04-21T13:17:52.000Z
4 :  2018-04-21T13:18:52.000Z
5 :  2018-04-21T13:19:52.000Z
6 :  2018-04-21T13:20:53.000Z
7 :  2018-04-21T13:21:52.000Z
8 :  2018-04-21T13:22:52.000Z
9 :  2018-04-21T13:23:52.000Z
...
...
741 :  2018-04-22T01:40:51.000Z
742 :  2018-04-22T01:41:51.000Z
743 :  2018-04-22T01:42:51.000Z
744 :  2018-04-22T01:43:51.000Z
745 :  2018-04-22T01:44:51.000Z

Parsed Disney take(10):


[[0, 600, 0],
 [0, 600, 55],
 [0, 600, 115],
 [0, 600, 175],
 [0, 600, 235],
 [0, 600, 295],
 [0, 600, 356],
 [0, 600, 415],
 [0, 600, 475],
 [0, 600, 535]]
```

```
...
Categorias:


{'ACCION': 4, 'BOTES': 5, 'FUTURO': 1, 'MONTANA': 2, 'MUSICAL': 0, 'NINOS': 3}



Actividades:


{'"""its a small world"""': 18,
 'Astro Orbiter': 1,
 'Big Thunder Mountain Railroad': 8,
 'Buzz Lightyears Space Ranger Spin': 13,
 'Dumbo the Flying Elephant': 19,
 'Enchanted Tales with Belle': 0,
 'Haunted Mansion': 2,
 'Jungle Cruise': 16,
 'Mad Tea Party': 21,
 'Mickeys PhilharMagic': 15,
 'Monsters, Inc. Laugh Floor': 7,
 'Peter Pans Flight': 20,
 'Pirates of the Caribbean': 17,
 'Prince Charming Regal Carrousel': 12,
 'Seven Dwarfs Mine Train ': 4,
 'Space Mountain': 3,
 'Splash Mountain': 10,
 'The Barnstormer': 6,
 'The Magic Carpets of Aladdin': 11,
 'The Many Adventures of Winnie the Pooh': 9,
 'Tomorrowland Speedway': 14,
 'Under the Sea - Journey of The Little Mermaid': 5}



Description:


[[0, 0, 3, 0],
 [1, 1, 4, 0],
 [2, 1, 5, 1],
 [3, 2, 10, 1],
 [4, 2, 7, 1],
 [5, 3, 2, 1],
 [6, 3, 3, 0],
 [7, 3, 4, 0],
```

```
 [8, 2, 6, 1],
 [9, 3, 1, 0],
 [10, 2, 9, 0],
 [11, 3, 2, 0],
 [12, 3, 0, 0],
 [13, 4, 6, 1],
 [14, 1, 2, 1],
 [15, 0, 4, 0],
 [16, 5, 2, 0],
 [17, 5, 6, 1],
 [18, 3, 1, 1],
 [19, 3, 0, 0],
 [20, 3, 1, 0],
 [21, 3, 2, 1]]
```

```
In [5]: args = [
            activitiesDict,
            categoriesDict,
            description,
            disneyDict,
            MAX_ACTIVITIES,
            MAX_CATEGORIES,
            MAX_MONTANA,
            MAX_NINOS,
            MAX_IMPRESCINDIBLES,
            True
        ]
        madeActivities, categoriesQuant, activitiesQuant = heuristica(args)
        print ""
        print "final score: ", getFinalScore(madeActivities)
        print ""
        print "QuantityResume: "
        display(getQuantityResume(activitiesQuant, activitiesDict))
        print ""
        print "Resume: "
        display(getResume(madeActivities, tookTimeDicc))
```

```
Time used: 0.141438961029 seconds
Cantidad de imprescindibles 11 de un total de  11
4 de  4 montañas
9 de  10 niños
Mickeys PhilharMagic:  2

final score:  139

QuantityResume:
```

```
[['Enchanted Tales with Belle', 2],
 ['Astro Orbiter', 2],
 ['Haunted Mansion', 3],
 ['Space Mountain', 1],
 ['Seven Dwarfs Mine Train ', 1],
 ['Under the Sea - Journey of The Little Mermaid', 2],
 ['The Barnstormer', 1],
 ['Monsters, Inc. Laugh Floor', 1],
 ['Big Thunder Mountain Railroad', 2],
 ['The Many Adventures of Winnie the Pooh', 0],
 ['Splash Mountain', 0],
 ['The Magic Carpets of Aladdin', 1],
 ['Prince Charming Regal Carrousel', 0],
 ['Buzz Lightyears Space Ranger Spin', 4],
 ['Tomorrowland Speedway', 2],
 ['Mickeys PhilharMagic', 2],
 ['Jungle Cruise', 4],
 ['Pirates of the Caribbean', 3],
 ['"""its a small world"""', 2],
 ['Dumbo the Flying Elephant', 0],
 ['Peter Pans Flight', 0],
 ['Mad Tea Party', 2],
 ['food', 1]]
```

Resume:

```
[['Haunted Mansion', '01:34:51', 15],
 ['Haunted Mansion', '01:19:51', 15],
 ['Monsters, Inc. Laugh Floor', '01:13:51', 5],
 ['Pirates of the Caribbean', '01:07:51', 5],
 ['Pirates of the Caribbean', '01:01:51', 5],
 ['Tomorrowland Speedway', '00:55:51', 5],
 ['Enchanted Tales with Belle', '00:36:53', 15],
 ['Under the Sea - Journey of The Little Mermaid', '23:31:50', 65],
 ['"""its a small world"""', '23:00:51', 30],
 ['Haunted Mansion', '22:40:50', 20],
 ['Jungle Cruise', '22:34:50', 5],
 ['Mickeys PhilharMagic', '22:13:50', 20],
 ['Mad Tea Party', '21:54:53', 10],
 ['Jungle Cruise', '21:44:50', 10],
 ['Astro Orbiter', '21:24:50', 20],
 ['Big Thunder Mountain Railroad', '21:09:50', 10],
 ['Tomorrowland Speedway', '20:42:51', 25],
 ['Jungle Cruise', '20:30:50', 10],
 ['Buzz Lightyears Space Ranger Spin', '20:15:49', 15],
```

```
['The Magic Carpets of Aladdin', '19:18:50', 55],
['Jungle Cruise', '19:01:49', 10],
['Enchanted Tales with Belle', '18:31:49', 30],
['Pirates of the Caribbean', '17:46:49', 45],
['Astro Orbiter', '17:25:49', 20],
['Mickeys PhilharMagic', '16:52:49', 30],
['"""its a small world"""', '16:37:49', 15],
['Under the Sea - Journey of The Little Mermaid', '15:18:50', 75],
['The Barnstormer', '14:58:48', 20],
['Buzz Lightyears Space Ranger Spin', '14:53:48', 5],
['food', '14:13:48', 40],
['Mad Tea Party', '14:07:48', 5],
['Buzz Lightyears Space Ranger Spin', '14:01:48', 5],
['Big Thunder Mountain Railroad', '13:51:48', 10],
['Seven Dwarfs Mine Train ', '13:31:48', 20],
['Buzz Lightyears Space Ranger Spin', '13:30:49', 0],
['Space Mountain', '13:14:57', 10]]
```

4 Cota superior

```
In [6]: def getMaxMin(aList, f):
            Max = f(aList[0])
            Min = f(aList[0])
            posMax = 0
            posMin = 0
            size = len(aList)
            for i in range(1, size):
                elem = f(aList[i])
                if (elem > Max):
                    Max = elem
                    posMax = i
                elif (elem < Min):
                    Min = elem
                    posMin = i
            return Min, Max, posMax, posMin
```

```
In [7]: # description[i] = [activity, category, points, imprescindible]
        # disneyDict[key] = [waittime, tooktime]

        print "MAX_TOOKTIME: ", MAX_TOOKTIME
        print "MAX_ACTIVITIES: ", MAX_ACTIVITIES
        print "MAX_TOOKTIME % MAX_ACTIVITIES: ", MAX_TOOKTIME/MAX_ACTIVITIES
        print ""

        upperBound_1 = 0
        for i in range(0, MAX_ACTIVITIES):
```

```
            upperBound_1 += description[i][2]*MAX_TOOKTIME
        print "upperBound_1: ", upperBound_1
        print ""

        upperBound_2 = 0
        for i in range(0, MAX_ACTIVITIES):
            upperBound_2 += description[i][2]*(32)
        print "upperBound_2: ", upperBound_2
        print ""

        upperBound_1 = 0
        firstTookTime, lastTookTime, NULL, NULL = getMaxMin(tookTimeDicc.keys(), lambda x: x)
        NULL, activityMaxPoints, pos, NULL = getMaxMin(description, lambda x: x[2])
        activity = description[pos][0]
        waittimeMin, NULL, NULL, NULL = getMaxMin(disneyDict[activity], lambda x: x[0])
        print "firstTookTime: ", firstTookTime
        print "lastTookTime: ", lastTookTime
        print "activityMaxPoints: ", activityMaxPoints
        print "waittime: ", waittimeMin
        print "activityMaxPoints* [(lastTookTime - firstTookTime) / waittimeMin]"
        print "upperBound_3: ", activityMaxPoints* (lastTookTime/waittimeMin)
```

```
MAX_TOOKTIME:  746
MAX_ACTIVITIES:  23
MAX_TOOKTIME % MAX_ACTIVITIES:  32

upperBound_1:  59680

upperBound_2:  2560

firstTookTime:  0
lastTookTime:  44994
activityMaxPoints:  10
waittime:  600
activityMaxPoints* [(lastTookTime - firstTookTime) / waittimeMin]
upperBound_3:  740
```