Universitat Autònoma de Barcelona, Module 01 Fall 2016
MSc in Bioinformatics

Algorithms: Assignments

Edgar G Nogales, 1441015

11/14/2016

1 Find a number in Sorted List

Input: A sorted list L of n numbers, and a number v. The list L can contain repeated values
Output: Position of first occurrence of v in L, or raise a value error if v is not found in L
Solution:

```
def find_binary(L, v):
        left=0
        right=len(L)
        if (L[0] == v): return 0;
        while ((right-left) > 1):
                m= (left+right)/2
                if (L[m] == v):
                        if (L[m-1] == v):
                                right= m
                        else:
                                return m
                if (L[m] < v):
                        left= m
                else:
                        right= m
        raise ValueError('Value is not in list')

L=[0,5,9,9,10,15,20];
print (find_binary(L,20))
```

2 Recursive version of Find maximum number in List

Input: A list L of n numbers. The list L can contain repeated values
Output: Maximum value in L, or raise a value error if L is empty (like built-in function max())
Solution:

```
def Max(list):
        if len(list)==0:
                raise ValueError('Empty list')
        elif len(list) == 1:
                return list[0]
        else:
                m = Max(list[1:])
                return m if m > list[0] else list[0]

L=[-5,-16,5,9,0];
print (Max(L))
```

3 Find a number in Sorted List, recursive version

Input: A sorted list L of n numbers, and a number v. The list L can contain repeated values
Output: Position of first occurrence of v in L, or raise a value error if v is not found in L
Solution:

```
def findB(L,V):
        N = len(L)
        if N == 0:
                raise ValueError ('Value is not a list')
        if N==1:
                if L[0]==V:
                        return 0
                raise ValueError ('Value is not in the list')
        m = N/2
        if V==L[m]:
                if V== L[m-1]:
                        return findB(L[:m],V)
                return m
        if V<L[m]:
                return findB(L[:m],V)
```

```
        return m+findB(L[m:],V)

L=[0,9,9,9,10,12]
print (findB(L,9))
```

4 Enumerating k-mers lexicographically

Input: A collection of at most 10 symbols defining an ordered alphabet, and a positive integer n.
Output: All strings of length n that can be formed from the alphabet, ordered lexicographically.

```
def alpha_combs(alphabet, n, acc='', res=[]):
    if n == 0:
        res.append(acc)
    else:
        for c in alphabet:
            alpha_combs(alphabet, n - 1, acc + c, res)
    return res

alphabet = ['A','T','C','G']
n=2
t= alpha_combs(alphabet,n)

for i in t:
        print i
#AA AT AC AG TA TT TC TG CA CT CC CG GA GT GC GG
```

5 Find k-mer in string with higher number of occurrences

```
def kmerOccurence(S,k):
        res=""
        data = dict()
        if (k<=1 or k>len(S)): #check el MENOS 1
                raise ValueError ('Value is not a list')
        for i in range(0, len(S)-k+1):
                for t in range(0, k):
                        res+=S[i+t]
                        #print S[i+t],"-----",i,"\t",t
                #print "*************",res
```

```
                if res in data:
                        data[res]+=1
                else:
                        data[res]=1
                res=""
                #print data
        return max(data, key=lambda i: data[i])

S="AATTATAT"
print (kmerOccurence(S,3))
```

6 Finding a Motif in DNA

Given two strings s and t, t is a substring of s if t is contained as a contiguous collection of symbols in s (as a result, t must be no longer than s). The position of a symbol in a string is the total number of symbols found to its left, including itself (e.g., the positions of all occurrences of 'U' in "AUGCUUCAGAAAGGUCUUACG" are 1, 4, 5, 14, 16, and 17). The symbol at position i of s is denoted by s[i]. A substring of s can be represented as s[j:k], where j and k represent the starting and ending positions of the substring in s; for example, if s = "AUGCUUCAGAAAGGUCUUACG", then s[1:4] = "UGCU". The location of a substring s[j:k] is its beginning position j; note that t will have multiple locations in s if it occurs more than once as a substring of s (see the Sample below).
Input: Two DNA strings s and t (each of length at most 1 Kbp).
Output: All locations of t as a substring of s.

```
seq = "GATATATGCATATACTT "
subs ="ATAT"
# Start with this value.
location = -1

# Loop while true.
while True:
    # Advance location by 1.
    location = seq.find(subs, location + 1)

    # Break if not found.
    if location == -1: break

    # Display result.
    print(location)
#1
#3
#9
```

7 Find minimum length path between N cities

Input: N cities + distance between cities
Output: a path traversing all cities with minimum length

```
import math
from itertools import permutations

def randList(N,a,b):
 import random
 return [random.randint(a,b) for i in range(0,N)]

def randCities(N,SZ):
 return zip(range(0,N), randList(N,0,SZ), randList(N,0,SZ))

def distance (x,y):
        return math.sqrt((x[1] - y[1])**2 + (x[2] - y[2])**2)

def bruteForce(points):
        return min([perm for perm in permutations(points)],
    key=total_distance)

def total_distance(points):
         return sum([distance(point, points[index + 1])
     for index, point in enumerate(points[:-1])])

lista=randCities(4,8)
print(lista)
print (total_distance(bruteForce(lista)))
print(bruteForce(lista))
```

8 Find minimum length path between N cities

Input: N cities + distance between cities
Output: a path traversing all cities with minimum length

```
import math
from itertools import permutations

def randList(N,a,b):
 import random
 return [random.randint(a,b) for i in range(0,N)]

def randCities(N,SZ):
 return zip(range(0,N), randList(N,0,SZ), randList(N,0,SZ))
```

```
def distance (x,y):
        return math.sqrt((x[1] - y[1])**2 + (x[2] - y[2])**2)

def heuristic(points):
        must_visit = points
        path = [points[0]]
        must_visit.remove(points[0])
        while must_visit:
                nearest = min(must_visit, key=lambda x:
        distance(path[-1], x))
                path.append(nearest)
                must_visit.remove(nearest)
        print (path)
        return path

def total_distance(points):
         return sum([distance(point, points[index + 1])
     for index, point in enumerate(points[:-1])])

lista=randCities(4,8)
print(lista)
print (total_distance(heuristic(lista)))
```