IE531 Programming\_Assignment3

Name: ChenYuhao

NetId: yuhaoc2

The plot for the Mean and Standard-Deviation of the running-time vs. n plots for different values of m is shown below:

图片包含 文字, 地图

描述已自动生成

The output of the code is shown in the following figure:

图片包含 文字, 户外

描述已自动生成

Plots of the slopes of the best-fit linear regressor as a function of m is shown below:

图片包含 文字, 地图

描述已自动生成

The time increment with increased pivot number is due to several factors. The first factor is the slowness of numpy.random.randint() as we generate the random pivots, when the array length is small, the generating time will take up more than 30%. The second factor is the compare and select time costed by a larger number of pivots, bringing m pivots to choose the one with the smallest length.

Therefore, more pivots included, the efficiency of the randomized selection algorithm doesn’t improve. There is no benefit for including multiple random pivot than using single pivot.

Appendix

Code:

#!/usr/bin/env python3

# coding: utf-8

# Experimentally determining the statistics of the running-time of

# picking the k-th smallest number in an unordered/unsorted list of numbers

# using randomly selected, m-many pivots; followed by a recursion on the

# shorter of the m-many sub-lists that contains the k-th smallest element

#

# IE531: Algorithms for Data Analytics

# Written by Prof. R.S. Sreenivas

#

import sys

import argparse

import random

import numpy as np

import time

import math

import matplotlib.pyplot as plt

sys.setrecursionlimit(3000)

def split\_into\_three(current\_array,pivot,k):#utility function for convenience follow sample code in Randomized\_Selection\_Rev.py

    Less\_than\_p = []

    Equal\_to\_p = []

    Greater\_than\_p = []

    for x in current\_array:

        if (x<pivot):

            Less\_than\_p.extend([x])

        if (x==pivot):

            Equal\_to\_p.extend([x])

        if (x>pivot):

            Greater\_than\_p.extend([x])

    if k<len(Less\_than\_p):

        return Less\_than\_p,0 #0 is the flag for lessthancase

    elif k>=len(Less\_than\_p)+len(Equal\_to\_p):

        return Greater\_than\_p,1  #1 is the flag for greater than case

    else:

        return Equal\_to\_p,2

def randomized\_select\_with\_multipe\_pivots (current\_array, k, no\_of\_pivots) :

    i=0

    pivot\_list=[]

    pivot\_arraylength={}

    pivot\_type={}

    pivot\_array={}

    if (len(current\_array) == 1) :

        return current\_array[0]

    if len(current\_array)<=no\_of\_pivots:

        pivot\_list=np.random.randint(min(current\_array)-1,max(current\_array)+1,no\_of\_pivots)

    if len(current\_array)>no\_of\_pivots:

        while i<=no\_of\_pivots:

            #print('min=',min(current\_array),'max=',max(current\_array))

            random\_num=np.random.randint(min(current\_array)-1,max(current\_array)+1)

            if random\_num not in pivot\_list:

                pivot\_list.extend([random\_num])

                i+=1

    for j in pivot\_list:

        array\_splitted,type\_of\_pivot=split\_into\_three(current\_array,j,k)

        pivot\_arraylength[j]=len(array\_splitted)#key is the pivot element, value is the length corresponding to the candidate subarrays

        pivot\_type[j]=type\_of\_pivot

        pivot\_array[j]=array\_splitted

        if type\_of\_pivot==2:

            return j

        #print('finish dict generation')

    min\_pivot=min(pivot\_arraylength,key=pivot\_arraylength.get)

    array\_next=pivot\_array[min\_pivot]

    if pivot\_type[min\_pivot]==0:

        return randomized\_select\_with\_multipe\_pivots(array\_next, k,no\_of\_pivots)

    elif pivot\_type[min\_pivot]==1:#array\_next is the greater than array

        return randomized\_select\_with\_multipe\_pivots(array\_next, k - (len(current\_array)-len(array\_next)),no\_of\_pivots)

    else:

            return min\_pivot

# Maximum #pivots

max\_no\_of\_pivots = 15

# Number of Trials

number\_of\_trials = 1000

# We are going to see if there is any observable difference in the slope of the Linear Regressor

# for the Mean (resp. Standard-Deviation) of the Running Time

# and the slope of standard-deviation-regressor as the number of pivots are increased

slope\_of\_mean\_regressor\_as\_a\_function\_of\_no\_of\_pivots = []

slope\_of\_std\_dev\_regressor\_as\_a\_function\_of\_no\_of\_pivots = []

# I am going to plot a lot of things

# I found the stuff here -- https://matplotlib.org/gallery/subplots\_axes\_and\_figures/figure\_title.html

# to be useful. Instead, I just used what got from here to get the subplots not to get squished down --

# https://stackoverflow.com/questions/41530975/set-size-of-subplot-in-matplotlib

fig = plt.figure(figsize=(8, 20))

# try #pivots = 1,2,3,4 and see if having more pivots is helping with the run-time

for number\_of\_pivots in range(1, max\_no\_of\_pivots+1) :

    # arrays containing mean- and std-dev of running time as a function of

    # array size starting from 100 to 4000 in steps of 100

    mean\_running\_time = []

    std\_dev\_running\_time = []

    mean\_generating\_time=[]

    # cycle through a bunch of array sizes, where "k" is randomly chosen

    for j in range(1, 40) :

        array\_size = 100\*j

        # let is pick k to be (close to) the median

        k = math.ceil(array\_size/2)

        # fill the array with random values

        my\_array = [random.randint(1,100\*array\_size) for \_ in range(array\_size)]

        # run a bunch of random trials and get the algorithm's running time

        running\_time = []

        generating\_time=[]

        for i in range(1, number\_of\_trials) :

            t1 = time.time()

            answer1 = randomized\_select\_with\_multipe\_pivots(my\_array,k-1,number\_of\_pivots)

            t2 = time.time()

            running\_time.extend([t2-t1])

            t3=time.time()

            pivot\_list=np.random.randint(min(my\_array)-1,max(my\_array)+1,number\_of\_pivots)

            t4=time.time()

            generating\_time.extend([t4-t3])

            # uncomment the lines below to verify the solution of randomized\_select\_with\_pivots

            answer2 = sort\_and\_select(my\_array, k)

            if (answer1 != answer2) :

                print ("Something went wrong")

                exit()

        mean\_running\_time.extend([np.mean(running\_time)])

        std\_dev\_running\_time.extend([np.std(running\_time)])

        mean\_generating\_time.extend([np.mean(generating\_time)])

    print('ratio of random number over total=',np.mean(mean\_generating\_time)/np.mean(mean\_running\_time))

    # linear fit (cf. https://docs.scipy.org/doc/numpy-1.15.0/reference/generated/numpy.polyfit.html)

    t = np.arange(100, 4000, 100)

    z1 = np.polyfit(t, mean\_running\_time, 1)

    p1 = np.poly1d(z1)

    z2 = np.polyfit(t, std\_dev\_running\_time, 1)

    p2 = np.poly1d(z2)

    print("#Pivots = ", number\_of\_pivots, "; Mean-Regressor's slope = ", z1[0], "; Std-Dev-Regressor's slope = ", z2[0])

    slope\_of\_mean\_regressor\_as\_a\_function\_of\_no\_of\_pivots.extend([z1[0]])

    slope\_of\_std\_dev\_regressor\_as\_a\_function\_of\_no\_of\_pivots.extend([z2[0]])

    # plot the mean and standard deviation of the running-time as a function of array-size

    axs = fig.add\_subplot(5, 3, number\_of\_pivots)

    plt.plot(t, mean\_running\_time, 'r', t, std\_dev\_running\_time, 'g', t, p1(t), 'r-', t, p2(t), 'g-')

    axs.set\_title('#Pivots =' + str(number\_of\_pivots))

plt.savefig("fig2.pdf", bbox\_inches='tight')

plt.show()

# plot the slope of the two regressors as a function of #pivots

x = [i for i in range(1, max\_no\_of\_pivots+1)]

plt.plot(x, slope\_of\_mean\_regressor\_as\_a\_function\_of\_no\_of\_pivots, 'r', x, slope\_of\_std\_dev\_regressor\_as\_a\_function\_of\_no\_of\_pivots, 'g')

plt.title('Linear Regressor Slope for Mean- and Std-Dev vs. #Pivots')

plt.xlabel('#Pivots')

plt.ylabel('Seconds/Length')

plt.savefig("fig1.pdf", bbox\_inches='tight')

plt.show()

# Checking if increasing the number of pivots is helping with the runtime in any significant manner...

z3 = np.polyfit(x, slope\_of\_mean\_regressor\_as\_a\_function\_of\_no\_of\_pivots, 1)

z4 = np.polyfit(x, slope\_of\_std\_dev\_regressor\_as\_a\_function\_of\_no\_of\_pivots, 1)

print("Sensitivity of the Slope of the Linear Regressor of the Mean to the #Pivots    = ", z3[0])

print("Sensitivity of the Slope of the Linear Regressor of the Std-Dev to the #Pivots = ", z4[0])

if len(current\_array)>no\_of\_pivots:

while i<=no\_of\_pivots:

random\_num=np.random.randint(min(current\_array)-1,max(current\_array)+1)

if random\_num not in pivot\_list:

pivot\_list.extend([random\_num])

i+=1

for j in pivot\_list:

array\_splitted,type\_of\_pivot=split\_into\_three(current\_array,j,k)

pivot\_arraylength[j]=len(array\_splitted)#key is the pivot element, value is the length corresponding to the candidate subarrays

pivot\_type[j]=type\_of\_pivot

pivot\_array[j]=array\_splitted

if type\_of\_pivot==2:

return j

#print('finish dict generation')

min\_pivot=min(pivot\_arraylength,key=pivot\_arraylength.get)

array\_next=pivot\_array[min\_pivot]

if pivot\_type[min\_pivot]==0:

return randomized\_select\_with\_multipe\_pivots(array\_next, k,no\_of\_pivots)

elif pivot\_type[min\_pivot]==1:#array\_next is the greater than array

return randomized\_select\_with\_multipe\_pivots(array\_next, k - (len(current\_array)-len(array\_next)),no\_of\_pivots)

else:

return min\_pivot