Университет ИТМО

Мобильные и сетевые технологии

**Лабораторная работа № 3**

***«Алгоритмы и структуры данных»***

Работу подготовил:

Кононов Степан

Группа К3140

Преподаватель: Татьяна Александровна Харьковская

Санкт-Петербург, 2021

1. Предоставить различные реализации quick\_sort, сравнить время работы

*"""Реализация quick\_sort"""*from random import randint  
from time import perf\_counter  
from prettytable import PrettyTable  
  
original\_array = [randint(-10\*\*4, 10\*\*4) for x in range(10\*\*5)]  
array\_version\_1 = original\_array[:]  
array\_version\_2 = original\_array[:]  
array\_version\_3 = original\_array[:]  
array\_version\_4 = original\_array[:]  
array\_len = len(original\_array)  
  
print("Progress: 0%")  
t\_start = perf\_counter()  
def quick\_sort(A, left, right):  
 if left < right:  
 x = A[left]  
 j = left  
 for i in range(left + 1, right):  
 if A[i] <= x:  
 j += 1  
 A[j], A[i] = A[i], A[j]  
 A[left], A[j] = A[j], A[left]  
 quick\_sort(A, left, j)  
 quick\_sort(A, j + 1, right)  
quick\_sort(array\_version\_1, 0, array\_len)  
time\_1 = perf\_counter() - t\_start  
  
print("Progress: 25%")  
t\_start = perf\_counter()  
def randomize\_quick\_sort(A, left, right):  
 if left < right:  
 k = randint(left, right-1)  
 A[left], A[k] = A[left], A[k]  
 x = A[left]  
 j = left  
 for i in range(left + 1, right):  
 if A[i] <= x:  
 j += 1  
 A[j], A[i] = A[i], A[j]  
 A[left], A[j] = A[j], A[left]  
 randomize\_quick\_sort(A, left, j)  
 randomize\_quick\_sort(A, j + 1, right)  
randomize\_quick\_sort(array\_version\_2, 0, array\_len)  
time\_2 = perf\_counter() - t\_start  
  
print("Progress: 50%")  
t\_start = perf\_counter()  
def quick\_sort\_equals(A, left, right):  
 if left < right:  
 x = A[left]  
 m1 = left  
 m2 = m1  
 for i in range(left + 1, right):  
 if A[i] < x:  
 m1 += 1  
 m2 += 1  
 A[m2], A[i] = A[i], A[m2]  
 A[m1], A[m2] = A[m2], A[m1]  
 elif A[i] == x:  
 m2 += 1  
 A[m2], A[i] = A[i], A[m2]  
 A[left], A[m1] = A[m1], A[left]  
 quick\_sort\_equals(A, left, m1)  
 quick\_sort\_equals(A, m2 +1, right)  
quick\_sort\_equals(array\_version\_3, 0, array\_len)  
time\_3 = perf\_counter() - t\_start  
  
print("Progress: 75%")  
t\_start = perf\_counter()  
def quick\_sort\_equals\_random(A, left, right):  
 if left < right:  
 k = randint(left, right-1)  
 A[left], A[k] = A[left], A[k]  
 x = A[left]  
 m1 = left  
 m2 = m1  
 for i in range(left + 1, right):  
 if A[i] < x:  
 m1 += 1  
 m2 += 1  
 A[m2], A[i] = A[i], A[m2]  
 A[m1], A[m2] = A[m2], A[m1]  
 elif A[i] == x:  
 m2 += 1  
 A[m2], A[i] = A[i], A[m2]  
 A[left], A[m1] = A[m1], A[left]  
 quick\_sort\_equals\_random(A, left, m1)  
 quick\_sort\_equals\_random(A, m2 +1, right)  
quick\_sort\_equals\_random(array\_version\_4, 0, array\_len)  
time\_4 = perf\_counter() - t\_start  
  
print("Progress: 100%")  
columns = ["Тип сортировки", "Время работы"]  
sort\_name = ["Обычная сортировка", "Рандомная сортировка", "Равная сортировка", "Рандомная равная сортировка"]  
sort\_time = [time\_1, time\_2, time\_3, time\_4]  
  
ans\_table = PrettyTable()  
ans\_table.add\_column(columns[0], sort\_name)  
ans\_table.add\_column(columns[1], sort\_time)  
print(ans\_table)  
  
'''  
print("Время работы обычной сортировки: %s секунд " % time\_1)  
print("Время работы рандомной сортировки: %s секунд " % time\_2)  
print("Время работы сортировки с учетом равных элементов: %s секунд " % time\_3)  
print("Время работы рандомной сортировки с учетом равных элементов: %s секунд " % time\_4)  
'''

1. Создать массив на котором quick\_sort выполняет наиболее большое ко-во действий

n = int(input())  
  
N = [x for x in range(1, n + 1)]  
  
for i in range(2, len(N)):  
 N[i], N[i // 2] = N[i // 2], N[i]  
  
print(N)

7. Дано n строк, выведите их порядок после k фаз цифровой сортировки.

from string import ascii\_lowercase  
  
n, k, length = map(int, input().split())  
  
A = ['bbb', 'aba', 'baa']  
rang = len(ascii\_lowercase)  
  
correspondence = dict()  
for i in range(len(ascii\_lowercase)):  
 correspondence[ascii\_lowercase[i]] = i + 1  
  
temp = dict()  
for i in range(len(A)):  
 temp[A[i]] = i + 1  
  
  
  
  
for i in range(length):  
 B = [[] for k in range(rang)] *#список длины range, состоящий из пустых списков* for x in A:  
 figure = x[-(i+1)]  
 B[correspondence[figure]].append(x)  
  
 A = []  
 for k in range(rang):  
 A = A + B[k]  
  
for i in A:  
 print(temp[i], end=' ')  
print()  
print(A)

8. В этой задаче, ваша цель - найти K ближайших точек к началу координат

среди данных n точек.

import math  
  
lenght, count = map(int, input().split())  
  
ans = []  
ar\_sort = False  
  
def calculate\_distance(x, y):  
 return math.sqrt(x\*\*2 + y\*\*2)  
  
for i in range(lenght):  
 X, Y = map(int, input().split())  
 if len(ans) < count:  
 ans.append([X, Y])  
 else:  
 if not ar\_sort:  
 ans = sorted(ans, key=lambda ar: calculate\_distance(ar[0], ar[1]))  
 ar\_sort = True  
 if calculate\_distance(X, Y) < calculate\_distance(ans[-1][0], ans[-1][1]):  
 ans[-1] = [X, Y]  
 min\_index = -2  
 while min\_index >= -len(ans) and calculate\_distance(X, Y) < calculate\_distance(ans[min\_index][0], ans[min\_index][1]):  
 ans[min\_index], ans[min\_index + 1] = ans[min\_index + 1], ans[min\_index]  
 min\_index -= 1  
print(\*ans)

9. В этой задаче, ваша цель - найти пару ближайших точек среди данных n точек

(между собой). Это базовая задача вычислительной геометрии, которая находит

применение в компьютерном зрении, систем управления трафиком.

import math  
import random  
  
  
def solution(x, y): *# x, y - списки координат точек типа n1 = (x[0], y[0])* a = list(zip(x, y)) *# Упаковываем в массив а кортежи координать точек* ax = sorted(a, key=lambda x: x[0]) *# ax - список точек отсортированные по X* ay = sorted(a, key=lambda x: x[1]) *# ay - список точек отсортированные по Y* p1, p2, mi = closest\_pair(ax, ay) *# Рекурсивный вызов* return mi  
  
def closest\_pair(ax, ay):  
 ln\_ax = len(ax)  
 if ln\_ax <= 3:  
 return brute(ax) *# При малом количестве точек вычисляем наивным алгоритмом* mid = ln\_ax // 2 *# Средняя линия деления* Lx = ax[:mid] *# Делим на две части (правая и левая)* Rx = ax[mid:]  
 *# Коодината линии деления по X* midpoint = ax[mid][0]  
  
 Ly = list()  
 Ry = list()  
 '''  
 Так как линия делит по X то Lx и Rx разделенены уже сразу, однако Ly и Ry так легко делить нельзя поэтому   
 делим их вручную  
 '''  
 for x in ay: *# Распределяем точки в левую и правую часть* if x[0] <= midpoint:  
 Ly.append(x)  
 else:  
 Ry.append(x)  
 *# Рекурсивный вызов для левой и правой части* (p1, q1, mi1) = closest\_pair(Lx, Ly)  
 (p2, q2, mi2) = closest\_pair(Rx, Ry)  
 *# Вычисляем минимальное расстояние среди левой и правой части  
 # P.S. mn - [[x0,y0][x1,y1]], p1/p2 и q1/q2 координаты ближайщих друг к другу точек с права и слева соответсвенно* if mi1 <= mi2:  
 d = mi1  
 mn = (p1, q1)  
 else:  
 d = mi2  
 mn = (p2, q2)  
  
  
 *# Вызываем функцию для нахождения точек возле средней линии* (p3, q3, mi3) = closest\_split\_pair(ax, ay, d, mn)  
  
 *# Сравниваем "центральное" расстояние с минимальным их правой/левой части* if d <= mi3:  
 return mn[0], mn[1], d  
 else:  
 return p3, q3, mi3  
  
*# Вычисление минимального расстояния просто через полный перебор*def brute(ax):  
 mi = dist(ax[0], ax[1])  
 p1 = ax[0]  
 p2 = ax[1]  
 ln\_ax = len(ax)  
 if ln\_ax == 2:  
 return p1, p2, mi  
 for i in range(ln\_ax-1):  
 for j in range(i + 1, ln\_ax):  
 if i != 0 and j != 1: *# Это уже вычислили (см на 5 строк выше)* d = dist(ax[i], ax[j])  
 if d < mi:  
 mi = d  
 p1, p2 = ax[i], ax[j]  
  
 return p1, p2, mi  
  
def dist(p1, p2):  
 return math.sqrt((p1[0] - p2[0]) \*\* 2 + (p1[1] - p2[1]) \*\* 2)  
  
*# Вычисляет мин рассотояние "срединных" точек (по разные строны от линии)*def closest\_split\_pair(p\_x, p\_y, delta, best\_pair):  
 ln\_x = len(p\_x)  
 mx\_x = p\_x[ln\_x // 2][0] *# Координата линии разделения  
 # Создаём подмассив точек не дальше чем дельта от  
 # средней точки на массиве с сортировкой по x* s\_y = [x for x in p\_y if mx\_x - delta <= x[0] <= mx\_x + delta]  
 best = delta *# Устанавливваем минимально расстояние* ln\_y = len(s\_y)  
 for i in range(ln\_y - 1):  
 for j in range(i+1, min(i + 7, ln\_y)):  
 p, q = s\_y[i], s\_y[j]  
 dst = dist(p, q)  
 if dst < best:  
 best\_pair = p, q  
 best = dst  
 return best\_pair[0], best\_pair[1], best  
  
  
print(solution([0, 3,7,7,7,], [0,4,6,5,4]))