МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
 РОССИЙСКОЙ ФЕДЕРАЦИИ  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Московский государственный университет геодезии и картографии»  
(МИИГАиК)  
Факультет геоинформатики и информационной безопасности  
Кафедра геоинформационных систем и технологий

**Лабораторная работа №7**  
**«Методы решения алгоритмических задач»**

Проверил: Выполнил:

Лебедев Е.Д. Студент группы: 2024-ФГиИБ-ПИ-1б

Николаев М.А.

Москва 2025

**Цель работы**

Выполнить предоставленные задачи на языке **Python.**

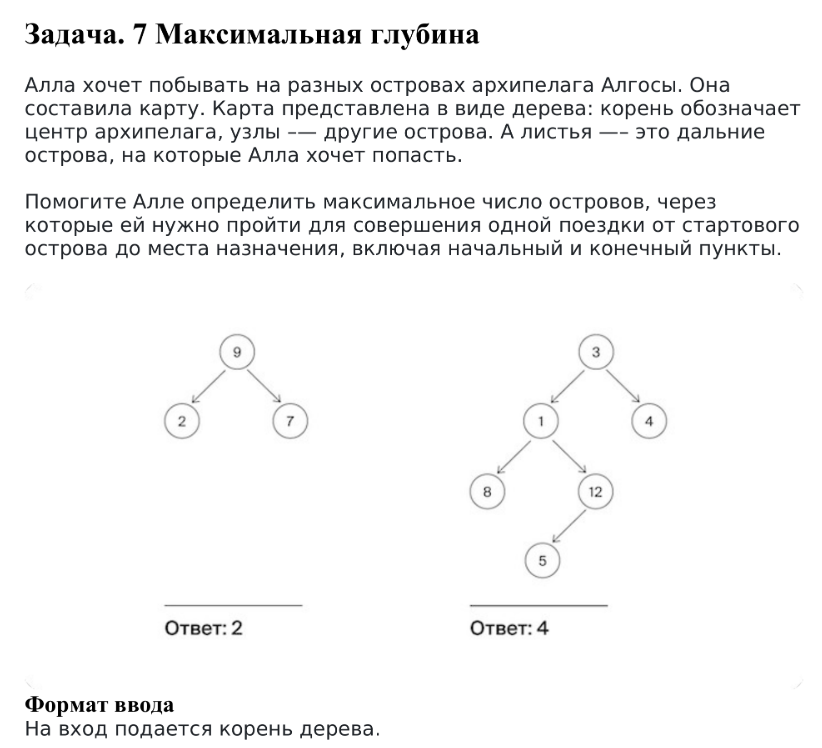
**Задание (вариант №19)**

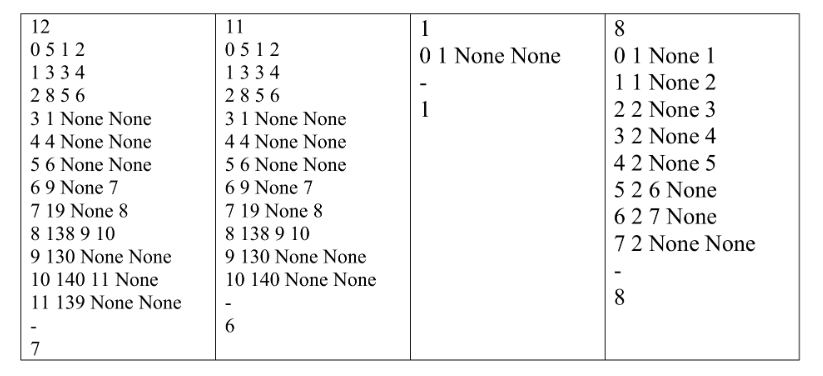
Выполнить 4 задания согласно варианту:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№ варианта** | **Деревья** | **Жадные алгоритмы и динамическое программирование** | **Разделяй и властвуй** | **Поиск с возвратом** |
| 19 | 7 | 2 | 5 | 3 |

**Анализ задания**

Задание на Деревья (№7)



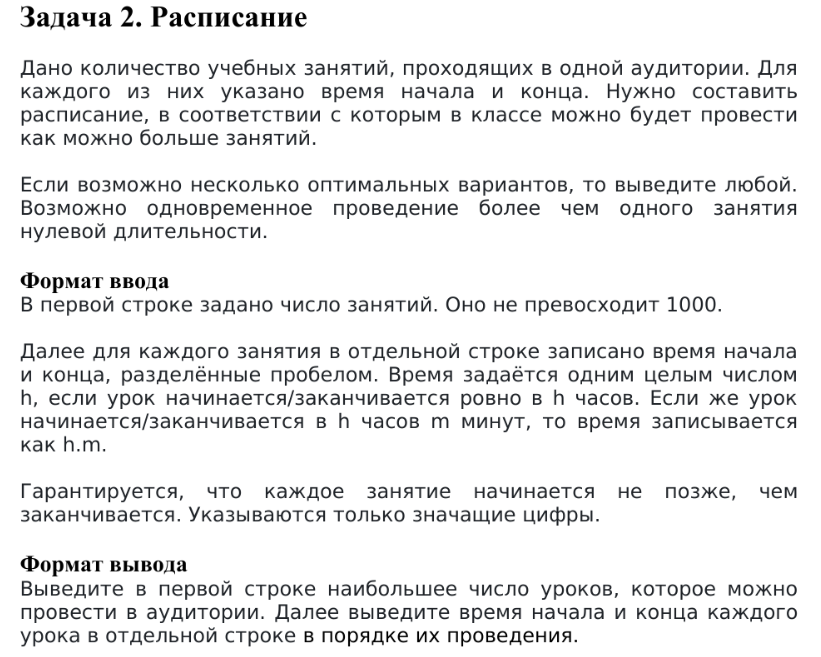
Входные данные: 

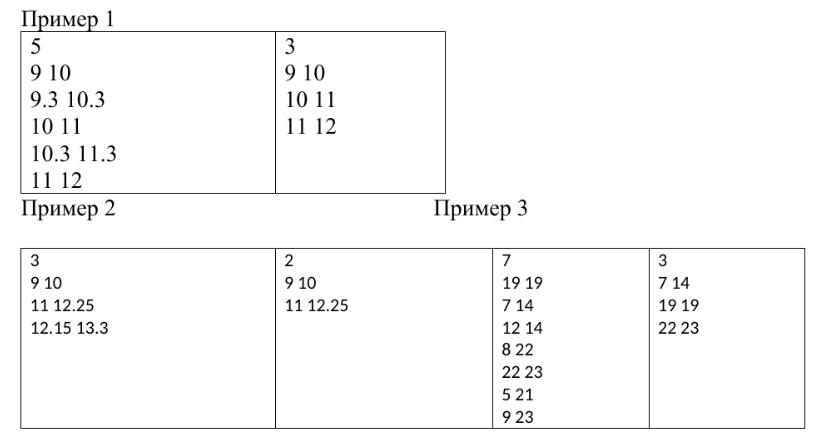
*Дерево (дерево рекурсий)* – это графический метод, который отображает разворачивание рекуррентного соотношения в систему рекурсивных вызовов. В этом дереве каждый узел представляет собой вызов функции, а ветви указывают на рекурсивные вызовы, которые делаются в процессе выполнения алгоритма.

Решение:

Создаём алгоритм проверки симметричности дерева относительно его центра. Для проверки симметричности нужно сравнить левое и правое поддерево на каждом уровне. Используем рекурсивный подход для обхода дерева и сравнения узлов. Дерево является анаграммой только если значения узлов и структура зеркально симметричны.

Задание на жадный алгоритм и динамическое программирования (№2)



Входные данные: 

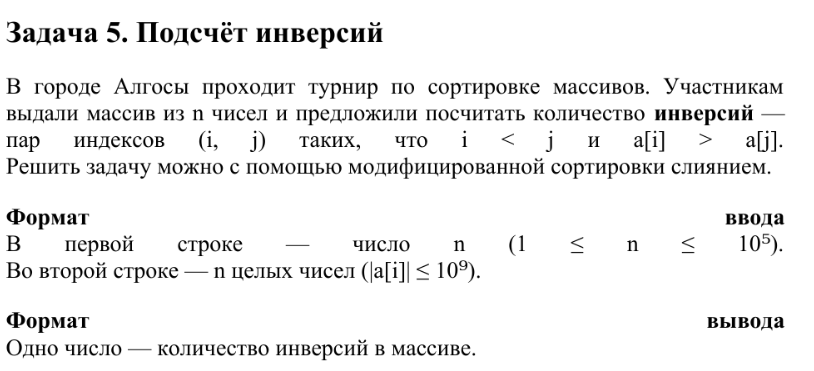
*Динамическое программирование* – это метод решения сложных задач путём разбиения их на более простые подзадачи и сохранение результатов решения этих подзадач для последующего использования.

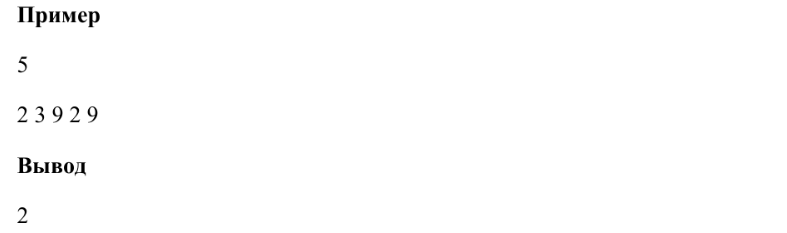
*Жадные алгоритмы* основаны на нахождении локально оптимальных решений на каждом этапе вычислений и допущении, что найденное таким образом итоговое решение будет оптимальным в глобальном смысле.

Решение:

Передаем рекурсивной функции корень дерева, проверяем есть ли дочерние узлы – если нет, добавляем значение корня в массив возможных чисел и возвращаем (т.е. конечный случай – возвращаем последнее число); иначе – проходим по каждому дочернему узлу, выполняем рек. функцию, введя узел как корень, после чего проходим по полученному массиву чисел и постепенно добавляем их в возвращаем массив, “склеивая” значение изначального корня (слева) с числом из массива (справа).

Задание на алгоритм разделяй и властвуй (№5)



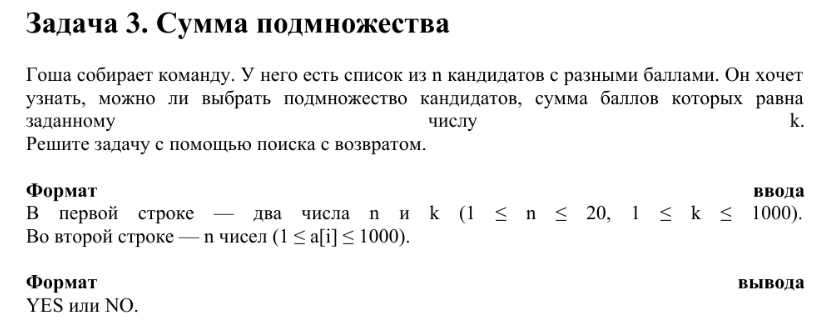
Входные данные: 

*Разделяй и властвуй* – это метод, разделяющий задачу на более мелкие подзадачи, независимо находящие решения для них и соединяющие результаты в решение изначальной задачи. Важнейший шаг в работе с алгоритмами “разделяй и властвуй” — это соединить решения подзадач в решение изначальной задачи

Решение:

Используя метод декомпозиции, разделяем задачи на более мелкие подзадачи. Подсчитываем количество инверсий (пар индексов (i, j) таких, что i < j и a[i] > a[j]) в массиве. При объединении массивов, с помощью сортировки слиянием, посчитываем количество инверсий.

Задание на алгоритм поиска с возвратом (№3)



Входные данные: 



*Алгоритм поиска с возвратом* относится к полным переборам, но у него есть особенность: если алгоритм понимает, что идёт по неверному пути, то все остальные варианты в этом пути тоже помечаются как неправильные и не рассматриваются. Ключевой элемент поиска с возвратом — рекурсия, то есть функция, которая вызывает сама себя.

Решение:

Поиск с возвратом в данной задаче реализуется следующим образом.

Мы последовательно перебираем все возможные комбинации чисел. Для каждого числа принимаем решение - берём его или нет.

Процесс перебора. На каждом шаге у нас есть два варианта: взять текущее число в сумму или пропустить текущее число.

Условия остановки. Успешное завершение – текущая сумма равна целевой неуспешное завершение: сумма превысила целевую, достигнут конец массива или все варианты перебраны.

**Когда нам удобно применять те или иные алгоритмы?**

* Деревья

Помимо организации поиска, деревья используют, когда разбирают математические выражения и компьютерные программы.

Еще их используют, чтобы хранить данные для алгоритмов сжатия, а также они лежат в основе других структур данных, например, очереди с приоритетом, кучи и словари.

* Жадные и динамические алгоритмы

Когда есть возможность разбить задачу на подзадачи, а также важна локальная оптимизация на каждом шагу.

* Метод разделяй и властвуй

Задачи с естественной рекурсивной структурой (например, сортировка слиянием).

* Поиск с возвратом

Поиск с возвратом чаще всего применяется в задачах комбинаторной оптимизации.

**Листинг программы**Ссылка на **GitHub**: <https://github.com/Ep1cMax/Second_Semester/tree/main/Works>

**HM7\_Trees.py**

import sys

from collections import deque

class TreeNode:

def \_\_init\_\_(self, val=0, left=None, right=None):

self.val = val

self.left = left

self.right = right

def buildTree():

lines = []

while True:

line = sys.stdin.readline().strip()

if line == '-':

break

if line:

lines.append(line)

if not lines:

return None

n = int(lines[0])

nodes = {}

for line in lines[1:n + 1]:

parts = line.split()

node\_id = int(parts[0])

value = int(parts[1])

nodes[node\_id] = TreeNode(value)

for line in lines[1:n + 1]:

parts = line.split()

node\_id = int(parts[0])

left\_id = int(parts[2]) if parts[2] != 'None' else None

right\_id = int(parts[3]) if parts[3] != 'None' else None

if left\_id is not None:

nodes[node\_id].left = nodes[left\_id]

if right\_id is not None:

nodes[node\_id].right = nodes[right\_id]

return nodes.get(0)

def maxDepth(root):

if not root:

return 0

queue = deque([(root, 1)])

max\_depth = 0

while queue:

node, depth = queue.popleft()

max\_depth = max(max\_depth, depth)

if node.left:

queue.append((node.left, depth + 1))

if node.right:

queue.append((node.right, depth + 1))

return max\_depth

root = buildTree()

print(maxDepth(root))

**HM7\_HungryDynam.py**

def select\_lessons(timetable):

timetable.sort(key=lambda x: (x[1], x[0]))

result = []

last\_end = 0.0

for start, end in timetable:

if start >= last\_end:

result.append((start, end))

last\_end = end

return result

print("Input number of lessons: ", end='')

lesson\_count = int(input())

lessons = []

for i in range(lesson\_count):

print(f"Input start/end time for lesson {i + 1}: ", end='')

start, end = map(float, input().split())

lessons.append((start, end))

selected = select\_lessons(lessons)

print(f"\n{len(selected)}")

for lesson in selected:

print(f"{lesson[0]} {lesson[1]}")

**HM7\_DivideAndConquer.py**

def merge\_sort\_plus\_count(arr, left, right):

if left >= right:

return 0

mid = (left + right) // 2

inv\_count = merge\_sort\_plus\_count(arr, left, mid)

inv\_count += merge\_sort\_plus\_count(arr, mid + 1, right)

inv\_count += merge\_and\_count(arr, left, mid, right)

return inv\_count

def merge\_and\_count(arr, left, mid, right):

left\_part = arr[left:mid + 1]

right\_part = arr[mid + 1:right + 1]

i = j = 0

k = left

inv\_count = 0

while i < len(left\_part) and j < len(right\_part):

if left\_part[i] <= right\_part[j]:

arr[k] = left\_part[i]

i += 1

else:

arr[k] = right\_part[j]

j += 1

inv\_count += (mid + 1) - (left + i)

k += 1

while i < len(left\_part):

arr[k] = left\_part[i]

i += 1

k += 1

while j < len(right\_part):

arr[k] = right\_part[j]

j += 1

k += 1

return inv\_count

def count\_inversions(arr):

return merge\_sort\_plus\_count(arr, 0, len(arr) - 1)

n = int(input())

arr = list(map(int, input().split()))

print(count\_inversions(arr))

**HM7\_Backtracking.py**

def form\_sum(candidates, target, curr\_sum=0, pos=0, used=None):

if used is None:

used = []

if curr\_sum == target:

return True

if curr\_sum > target or pos >= len(candidates):

return False

if form\_sum(candidates, target, curr\_sum + candidates[pos],

pos + 1, used + [candidates[pos]]):

return True

return form\_sum(candidates, target, curr\_sum, pos + 1, used)

n, k = map(int, input().split())

candidates = list(map(int, input().split()))

result = form\_sum(candidates, k)

print("YES" if result else "NO")

**Результаты работы программы:**

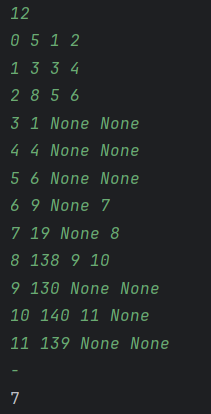


Рис. 1. – Работа алгоритма на деревья.

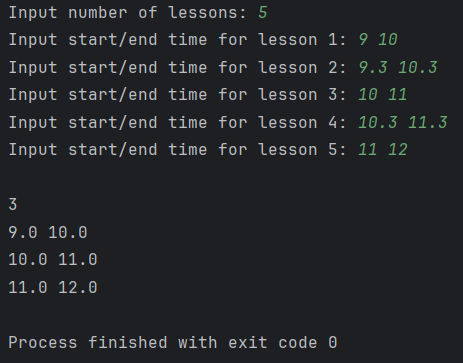


Рис. 2. – Работа жадных алгоритмов и динамического программирования.

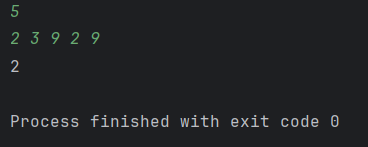


Рис. 3. – Работа алгоритма разделяй и властвуй.

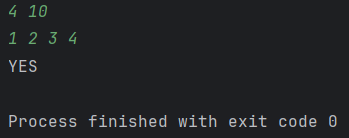


Рис. 4. – Работа алгоритма на поиск с возвратом