

Санкт-Петербургский Национальный Исследовательский Университет
Информационных Технологий, Механики и Оптики

Кафедра Систем Управления и Информатики

Лабораторная работа №8

Выполнил(и):

Доценников Н.А.

Проверил:

Мусаев А.А.

Санкт-Петербург,
2025

Введение

В данной лабораторной работе требуется разработать оригинальные задачи по ключевым темам алгоритмов. Для каждой задачи необходимо сформулировать условие, предложить решение и обосновать выбор метода. Цель работы – закрепить навыки постановки и анализа алгоритмических задач, а также демонстрация практического применения различных методов.

Алгоритмы сортировки

Задача 1.

Условие

В одном королевстве существует Гильдия Равновесия: в неё принимают только тройки героев, чьи силы в сумме равны нулю. Даны силы героев (могут повторяться). Найти все уникальные тройки, дающие сумму 0.

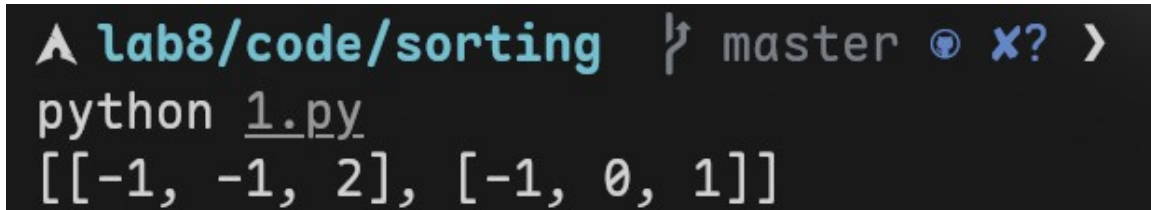
Решение

Отсортируем массив и переберём первый элемент тройки. Для оставшегося подмассива используем два указателя, сдвигая их в зависимости от текущей суммы. Для устранения дубликатов пропускаем повторы первого и второго элементов. Сложность $O(n^2)$.

```
def three_sum(nums):
    res = []
    nums.sort()
    for i in range(len(nums)):
        if i > 0 and nums[i] == nums[i-1]:
            continue
        j = i + 1
        k = len(nums) - 1
        while j < k:
            s = nums[i] + nums[j] + nums[k]
            if s > 0:
                k -= 1
            elif s < 0:
                j += 1
            else:
                res.append([nums[i], nums[j], nums[k]])
                j += 1
                while j < k and nums[j] == nums[j-1]:
```

```
        j += 1
    return res

if __name__ == "__main__":
    nums = [-1, 0, 1, 2, -1, -4]
    print(three_sum(nums))
```

A terminal window with a dark background. The prompt is 'lab8/code/sorting' with a blue arrow icon. The user has entered 'python 1.py'. The output is '[[-1, -1, 2], [-1, 0, 1]]'.

```
lab8/code/sorting master x? >
python 1.py
[[ -1, -1, 2], [ -1, 0, 1]]
```

Рисунок 1 – Скриншот работы кода

Задача 2.

Условие

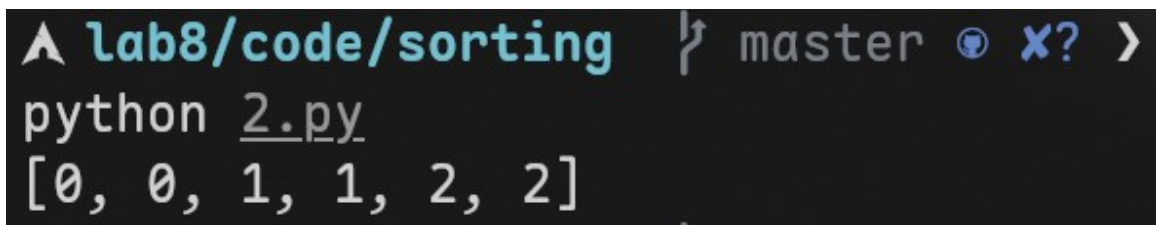
Даны полоски ткани трёх цветов: 0, 1, 2. Требуется упорядочить массив по возрастанию без вспомогательной памяти и без готовых функций сортировки.

Решение

Используем три указателя: границы для 0 и 2 и текущую позицию. При встрече 0 меняем с началом, при 2 — с концом, при 1 — продвигаем текущую позицию. Однопроходный алгоритм, $O(n)$ по времени и $O(1)$ по памяти.

```
def sort_colors(nums):
    low = 0
    mid = 0
    high = len(nums) - 1
    while mid <= high:
        if nums[mid] == 0:
            nums[low], nums[mid] = nums[mid], nums[low]
            low += 1
            mid += 1
        elif nums[mid] == 1:
            mid += 1
        else:
            nums[mid], nums[high] = nums[high], nums[mid]
            high -= 1

if __name__ == "__main__":
    nums = [2, 0, 2, 1, 1, 0]
    sort_colors(nums)
    print(nums)
```

A terminal window with a dark background. The prompt is 'lab8/code/sorting' in cyan, followed by 'master' in grey, a GitHub icon, 'x?' in blue, and a right arrow. The command 'python 2.py' is entered, and the output '[0, 0, 1, 1, 2, 2]' is displayed.

```
lab8/code/sorting master x? >  
python 2.py  
[0, 0, 1, 1, 2, 2]
```

Рисунок 2 – Скриншот работы кода

Задача 3.

Условие

Дан массив чисел. Переставить элементы так, чтобы получилась зигзагообразная последовательность: $\text{nums}[0] < \text{nums}[1] > \text{nums}[2] < \text{nums}[3] > \dots$.

Решение

Отсортируем массив, разделим пополам и развернём обе половины. Заполним чётные позиции элементами из левой половины, нечётные — из правой. Это даёт требуемое чередование. Сложность $O(n \log n)$.

```
def wiggle_sort(nums):
    nums.sort()
    mid = (len(nums) + 1) // 2
    left = nums[:mid][::-1]
    right = nums[mid:][::-1]
    nums[::2] = left
    nums[1::2] = right

if __name__ == "__main__":
    nums = [1, 5, 1, 1, 6, 4]
    wiggle_sort(nums)
    print(nums)
```

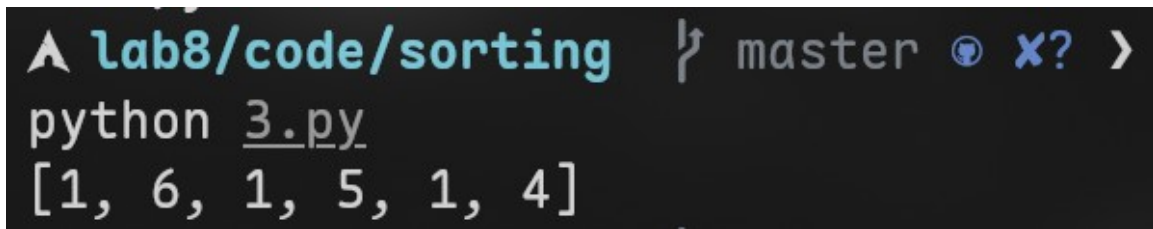


Рисунок 3 – Скриншот работы кода

Алгоритмы поиска

Задача 1.

Условие

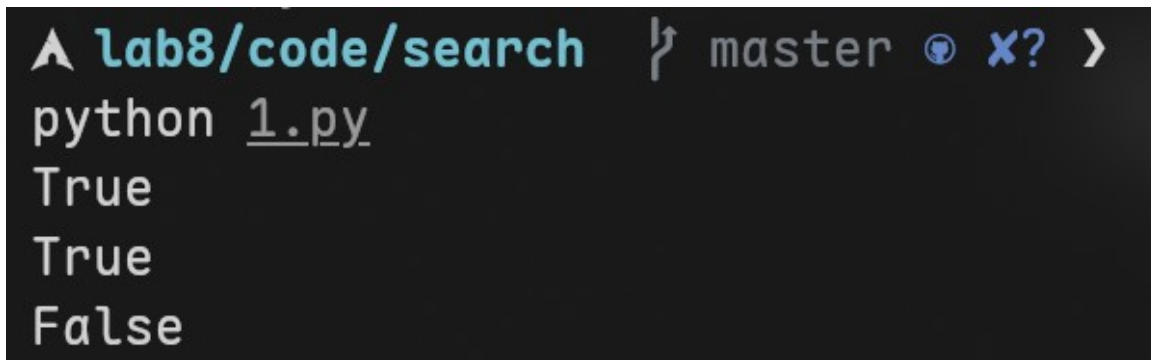
Дана прямоугольная решётка символов и слово. Можно ходить по соседним клеткам по вертикали и горизонтали, клетку можно использовать не более одного раза. Нужно определить, существует ли путь, формирующий слово.

Решение

Используем поиск в глубину с возвратом. Запускаем DFS из каждой клетки, совпадающей с первой буквой, помечая клетки как посещённые на текущем пути. Сложность $O(m \cdot n \cdot 4^L)$, где L — длина слова.

```
def exist(board, word):
    rows, cols = len(board), len(board[0])
    visited = set()
    def dfs(r, c, k):
        if k == len(word):
            return True
        if r < 0 or r >= rows or c < 0 or c >= cols or (r, c) in visited or board[r][c] != word[k]:
            return False
        visited.add((r, c))
        ok = dfs(r+1, c, k+1) or dfs(r-1, c, k+1) or dfs(r, c+1, k+1) or dfs(r, c-1, k+1)
        visited.remove((r, c))
        return ok
    for r in range(rows):
        for c in range(cols):
            if dfs(r, c, 0):
                return True
    return False
```

```
if __name__ == "__main__":  
    board = [['A','B','C','E'], ['S','F','C','S'], ['A','D','E','E']]  
    print(exist(board, "ABCCED"))  
    print(exist(board, "SEE"))  
    print(exist(board, "ABCB"))
```



```
lab8/code/search master x? >  
python 1.py  
True  
True  
False
```

Рисунок 1 – Скриншот работы кода

Задача 2.

Условие

Дан массив, отсортированный по возрастанию и повёрнутый на неизвестный сдвиг, и значение target. Найти индекс target за $O(\log n)$, иначе вернуть -1.

Решение

Модифицируем бинарный поиск: на каждом шаге одна половина упорядочена. Проверяем, лежит ли target в упорядоченной половине, и сужаем поиск соответствующим образом.

```
def search_rotated(nums, target):
    l, r = 0, len(nums) - 1
    while l <= r:
        m = (l + r) // 2
        if nums[m] == target:
            return m
        if nums[m] >= nums[l]:
            if nums[l] <= target < nums[m]:
                r = m - 1
            else:
                l = m + 1
        else:
            if nums[m] < target <= nums[r]:
                l = m + 1
            else:
                r = m - 1
    return -1

if __name__ == "__main__":
    print(search_rotated([4,5,6,7,0,1,2], 0))
    print(search_rotated([4,5,6,7,0,1,2], 3))
```

```
^ lab8/code/search ↵ master x? >  
python 2.py  
4  
-1
```

Рисунок 2 – Скриншот работы кода

Задача 3.

Условие

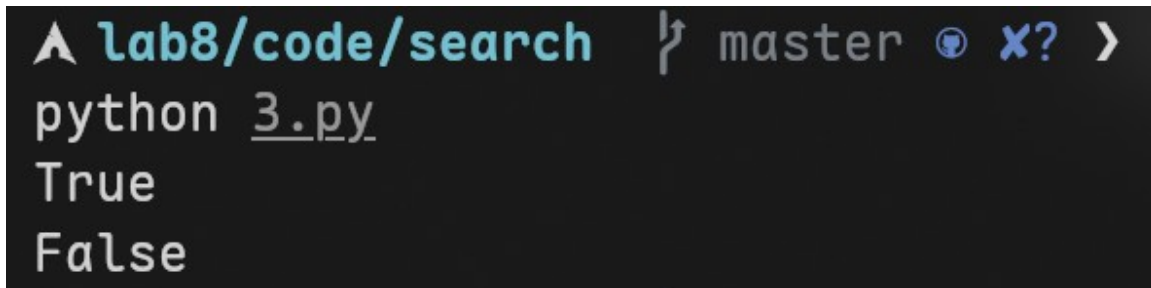
Дана матрица, где каждая строка отсортирована по возрастанию, а первый элемент строки строго больше последнего элемента предыдущей строки. Определить, содержит ли матрица `target`, за $O(\log(m \cdot n))$.

Решение

Сначала бинарным поиском находим строку, в диапазон которой попадает `target`, затем выполняем бинарный поиск по этой строке. Итого $O(\log m + \log n)$.

```
def search_matrix(matrix, target):
    if not matrix or not matrix[0]:
        return False
    top, bot = 0, len(matrix) - 1
    while top <= bot:
        mid = (top + bot) // 2
        if matrix[mid][0] <= target <= matrix[mid][-1]:
            row = mid
            break
        if target < matrix[mid][0]:
            bot = mid - 1
        else:
            top = mid + 1
    else:
        return False
    l, r = 0, len(matrix[row]) - 1
    while l <= r:
        m = (l + r) // 2
        if matrix[row][m] == target:
            return True
        if matrix[row][m] < target:
            l = m + 1
```

```
    else:  
        r = m - 1  
    return False  
  
if __name__ == "__main__":  
    matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]]  
    print(search_matrix(matrix, 3))  
    print(search_matrix(matrix, 13))
```



```
lab8/code/search master x? >  
python 3.py  
True  
False
```

Рисунок 3 – Скриншот работы кода

Хэширование

Задача 1.

Условие

Реализовать кэш LRU с операциями `get(key)` и `put(key, value)` за $O(1)$. При переполнении удаляется элемент, который дольше всех не использовался.

Решение

Используем хэш-таблицу для доступа по ключу и двусвязный список для поддержания порядка использования. Операции `get/put` перемещают узел в конец списка. При превышении вместимости удаляется головной узел.

```
class Node:
```

```
    def __init__(self, key, val):
        self.key = key
        self.val = val
        self.prev = None
        self.next = None
```

```
class LRUCache:
```

```
    def __init__(self, capacity: int):
        self.cap = capacity
        self.cache = {}
        self.left = Node(0, 0)
        self.right = Node(0, 0)
        self.left.next = self.right
        self.right.prev = self.left
```

```
    def remove(self, node):
        p, n = node.prev, node.next
        p.next = n
        n.prev = p
```

```

def insert(self, node):
    p, n = self.right.prev, self.right
    p.next = node
    node.prev = p
    node.next = n
    n.prev = node

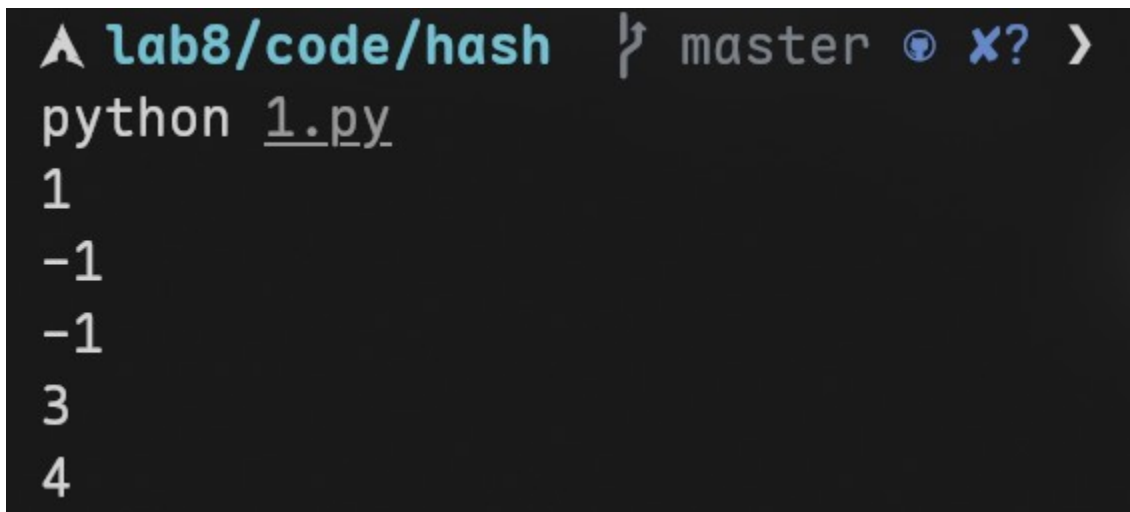
def get(self, key: int) -> int:
    if key not in self.cache:
        return -1
    node = self.cache[key]
    self.remove(node)
    self.insert(node)
    return node.val

def put(self, key: int, value: int) -> None:
    if key in self.cache:
        self.remove(self.cache[key])
    node = Node(key, value)
    self.cache[key] = node
    self.insert(node)
    if len(self.cache) > self.cap:
        lru = self.left.next
        self.remove(lru)
        del self.cache[lru.key]

if __name__ == "__main__":
    lru = LRUCache(2)
    lru.put(1, 1)
    lru.put(2, 2)

```

```
print(lru.get(1))  
lru.put(3, 3)  
print(lru.get(2))  
lru.put(4, 4)  
print(lru.get(1))  
print(lru.get(3))  
print(lru.get(4))
```



```
lab8/code/hash master x? >  
python 1.py  
1  
-1  
-1  
3  
4
```

Рисунок 1 – Скриншот работы кода

Задача 2.

Условие

Реализовать префиксное дерево с операциями `insert(word)`, `search(word)`, `startsWith(prefix)`.

Решение

В каждой вершине хранится словарь переходов и флаг конца слова. Вставка и поиск выполняются по символам.

```
class TrieNode:
    def __init__(self):
        self.children = {}
        self.is_end = False

class Trie:
    def __init__(self):
        self.root = TrieNode()

    def insert(self, word):
        node = self.root
        for ch in word:
            if ch not in node.children:
                node.children[ch] = TrieNode()
            node = node.children[ch]
        node.is_end = True

    def search(self, word):
        node = self.root
        for ch in word:
            if ch not in node.children:
                return False
            node = node.children[ch]
```

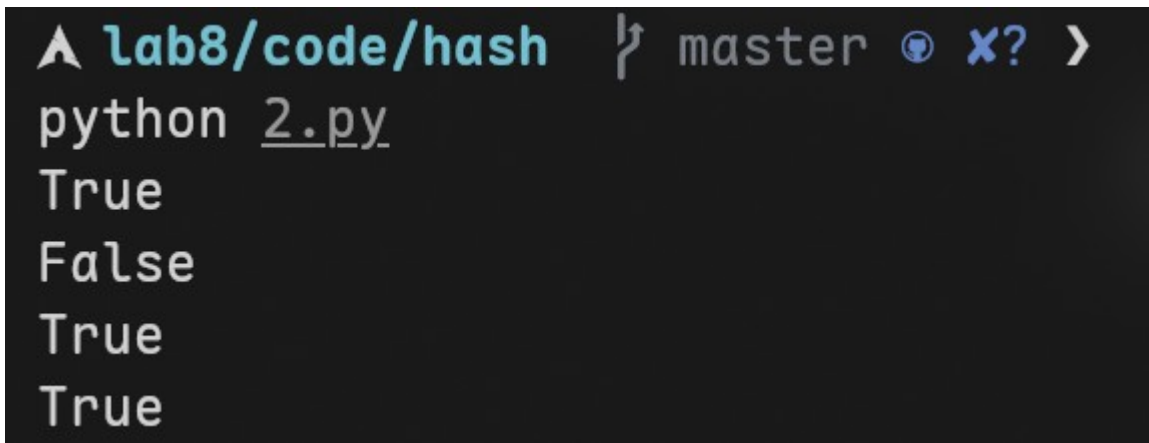
```

    return node.is_end

def startsWith(self, prefix):
    node = self.root
    for ch in prefix:
        if ch not in node.children:
            return False
        node = node.children[ch]
    return True

if __name__ == "__main__":
    t = Trie()
    t.insert("apple")
    print(t.search("apple"))
    print(t.search("app"))
    print(t.startsWith("app"))
    t.insert("app")
    print(t.search("app"))

```



```

^ lab8/code/hash  master  x? >
python 2.py
True
False
True
True

```

Рисунок 2 – Скриншот работы кода

Задача 3.

Условие

Спроектировать мини-Twitter: `postTweet(userId, tweetId)`, `getNewsFeed(userId)` возвращает до 10 последних твитов пользователя и тех, на кого он подписан, `follow/unfollow`.

Решение

Храним списки твитов на пользователя с убывающими временными метками и множество подписок. Ленту формируем с помощью кучи по меткам, извлекая до 10 самых новых.

```
import heapq
```

```
class Twitter:
```

```
    def __init__(self):
```

```
        self.time = 0
```

```
        self.tweets = {}
```

```
        self.follows = {}
```

```
    def postTweet(self, userId, tweetId):
```

```
        self.time += 1
```

```
        if userId not in self.tweets:
```

```
            self.tweets[userId] = []
```

```
        self.tweets[userId].append((-self.time, tweetId))
```

```
    def getNewsFeed(self, userId):
```

```
        heap = []
```

```
        if userId in self.tweets:
```

```
            heap.extend(self.tweets[userId][-10:])
```

```
        if userId in self.follows:
```

```
            for v in self.follows[userId]:
```

```
                if v in self.tweets:
```

```

        heap.extend(self.tweets[v][-10:])
    heapq.heapify(heap)
    res = []
    while heap and len(res) < 10:
        res.append(heapq.heappop(heap)[1])
    return res

def follow(self, followerId, followeeId):
    if followerId == followeeId:
        return
    if followerId not in self.follows:
        self.follows[followerId] = set()
    self.follows[followerId].add(followeeId)

def unfollow(self, followerId, followeeId):
    if followerId in self.follows and followeeId in self.follows[followerId]:
        self.follows[followerId].remove(followeeId)

if __name__ == "__main__":
    tw = Twitter()
    tw.postTweet(1, 5)
    print(tw.getNewsFeed(1))
    tw.follow(1, 2)
    tw.postTweet(2, 6)
    print(tw.getNewsFeed(1))
    tw.unfollow(1, 2)
    print(tw.getNewsFeed(1))

```

```
^ lab8/code/hash ↵ master 🔍 x? >
python 3.py
[5]
[6, 5]
[5]
```

Рисунок 3 – Скриншот работы кода

Жадные алгоритмы

Задача 1.

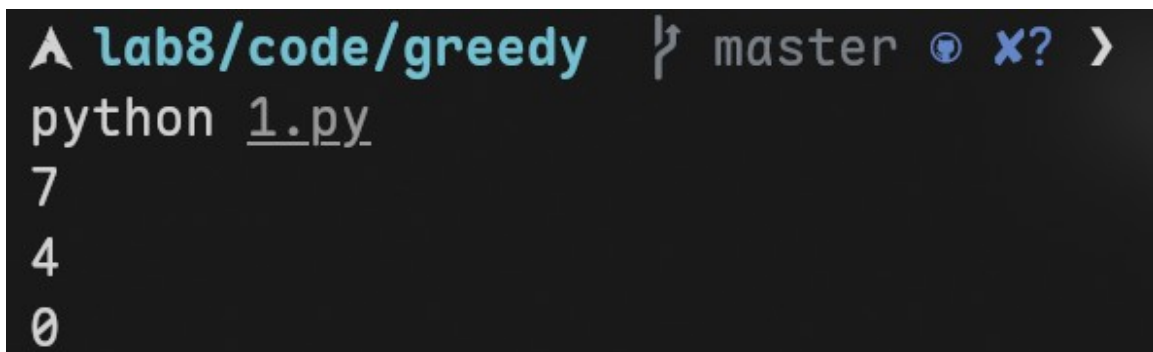
Условие

Даны цены на товар по дням. Можно совершать любые сделки покупки и продажи, но одновременно держать не более одной позиции. Найти максимальную прибыль.

Решение

Жадный принцип: суммируем все положительные приращения цен между соседними днями. Это эквивалентно разбиению на локальные минимумы и максимумы и даёт оптимум при множественных сделках.

```
def max_profit(prices):  
    profit = 0  
    for i in range(1, len(prices)):  
        if prices[i] > prices[i-1]:  
            profit += prices[i] - prices[i-1]  
    return profit  
  
if __name__ == "__main__":  
    print(max_profit([7,1,5,3,6,4]))  
    print(max_profit([1,2,3,4,5]))  
    print(max_profit([7,6,4,3,1]))
```



```
lab8/code/greedy master x? >  
python 1.py  
7  
4  
0
```

Рисунок 1 – Скриншот работы кода

Задача 2.

Условие

Заданы массивы `gas` и `cost` для кольцевого маршрута заправок. Нужно определить индекс заправки, с которой можно объехать круг, или `-1`, если это невозможно.

Решение

Если сумма `gas` меньше суммы `cost`, решения нет. Иначе идём по станциям, поддерживая текущий баланс; когда баланс падает ниже нуля, переносим старт на следующую станцию. В конце текущий старт — ответ.

```
def can_complete_circuit(gas, cost):
    if sum(gas) < sum(cost):
        return -1
    start = 0
    bal = 0
    for i in range(len(gas)):
        bal += gas[i] - cost[i]
        if bal < 0:
            start = i + 1
            bal = 0
    return start

if __name__ == "__main__":
    print(can_complete_circuit([1,2,3,4,5], [3,4,5,1,2]))
    print(can_complete_circuit([2,3,4], [3,4,3]))
```

```
^ lab8/code/greedy ↵ master Ⓢ x? >  
python 2.py  
3  
-1
```

Рисунок 2 – Скриншот работы кода

Задача 3.

Условие

Дан массив неотрицательных целых чисел. Требуется упорядочить их так, чтобы при конкатенации получить максимально возможное число. Вернуть результат строкой.

Решение

Жадный критерий сравнения: для двух строк *a* и *b* выбрать порядок по сравнению *b+a* и *a+b*. Сортируем по этому правилу и склеиваем. Если первый символ результата '0', ответ — '0'.

```
from functools import cmp_to_key
```

```
def largest_number(nums):
    arr = list(map(str, nums))
    def cmp(a, b):
        if a+b < b+a:
            return 1
        if a+b > b+a:
            return -1
        return 0
    arr.sort(key=cmp_to_key(cmp))
    res = ''.join(arr)
    return '0' if res[0] == '0' else res

if __name__ == "__main__":
    print(largest_number([10, 2]))
    print(largest_number([3,30,34,5,9]))
    print(largest_number([0,0]))
```

```
lab8/code/greedy master x? >
python 3.py
210
9534330
0
```

Рисунок 3 – Скриншот работы кода

Динамическое программирование

Задача 1.

Условие

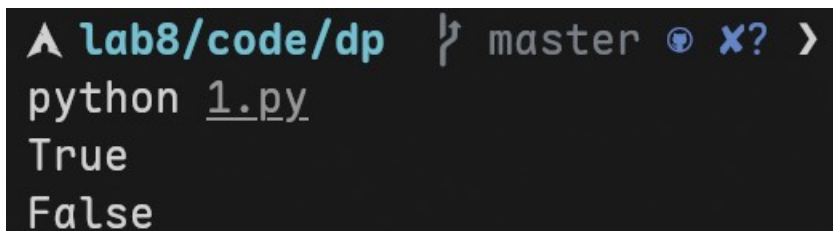
Дана строка s без пробелов и словарь `wordDict`. Нужно определить, можно ли разбить s на последовательность слов из словаря (слова можно использовать многократно).

Решение

Пусть $dp[i]$ — истина, если префикс $s[:i]$ разбивается на слова из словаря. Для каждого i перебираем слова w и проверяем $dp[i-\text{len}(w)]$ и совпадение $s[i-\text{len}(w):i]$ с w . Ответ $dp[\text{len}(s)]$. Сложность $O(N \cdot M \cdot L)$.

```
def word_break(s, wordDict):
    dp = [True] + [False] * len(s)
    for i in range(1, len(s) + 1):
        for w in wordDict:
            j = i - len(w)
            if j >= 0 and dp[j] and s[j:i] == w:
                dp[i] = True
                break
    return dp[-1]

if __name__ == "__main__":
    print(word_break("applepenapple", ["apple", "pen"]))
    print(word_break("catsanddog", ["cats", "dog", "sand", "and", "cat"]))
```



```
lab8/code/dp master x? >
python 1.py
True
False
```

Рисунок 1 – Скриншот работы кода

Задача 2.

Условие

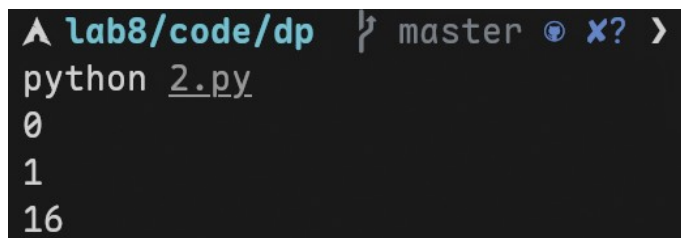
Задано число n . Скрытое число в диапазоне $[1..n]$. За неверную попытку k платим k монет. Нужно вычислить минимальную сумму монет, гарантирующую угадывание в худшем случае.

Решение

$dp[i][j]$ — минимальная стоимость угадывания числа в $[i..j]$. Для опорной точки p стоимость равна $p + \max(dp[i][p-1], dp[p+1][j])$. Берём минимум по p . Заполняем по длине отрезка. Сложность $O(n^3)$.

```
def min_cost_to_win(n):
    dp = [[0] * (n + 2) for _ in range(n + 2)]
    for length in range(2, n + 1):
        for i in range(1, n - length + 2):
            j = i + length - 1
            best = 10**9
            for p in range(i, j + 1):
                left = dp[i][p - 1] if p > i else 0
                right = dp[p + 1][j] if p < j else 0
                best = min(best, p + max(left, right))
            dp[i][j] = best
    return dp[1][n]

if __name__ == "__main__":
    print(min_cost_to_win(1))
    print(min_cost_to_win(2))
    print(min_cost_to_win(10))
```



```
lab8/code/dp master x? >
python 2.py
0
1
16
```

Рисунок 2 – Скриншот работы кода

Задача 3.

Условие

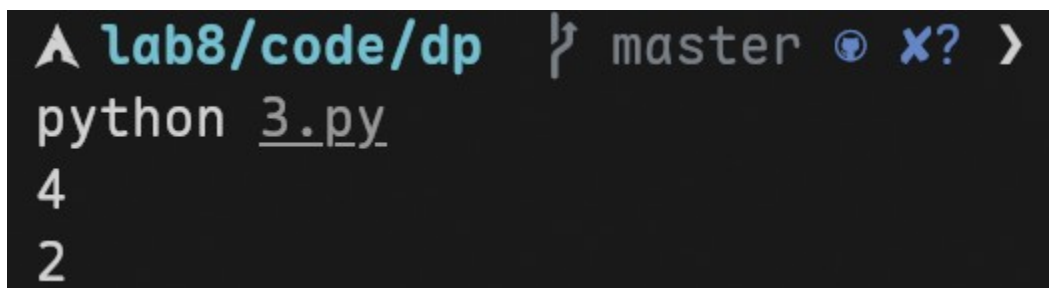
Даны бинарные строки `strs` и ограничения `m` по нулям и `n` по единицам. Найти максимальное число строк, которые можно выбрать, не превышая лимиты.

Решение

Классический 0/1-рюкзак по двум ресурсам. `dp[i][j]` — максимум строк при лимитах `i` нулей и `j` единиц. Для каждой строки с подсчитанными `zeros` и `ones` обновляем `dp` в обратном порядке. Сложность $O(l \cdot m \cdot n)$.

```
def find_max_form(strs, m, n):
    dp = [[0] * (n + 1) for _ in range(m + 1)]
    for s in strs:
        z = s.count('0')
        o = len(s) - z
        for i in range(m, z - 1, -1):
            for j in range(n, o - 1, -1):
                dp[i][j] = max(dp[i][j], dp[i - z][j - o] + 1)
    return dp[m][n]

if __name__ == "__main__":
    print(find_max_form(["10", "0001", "111001", "1", "0"], 5, 3))
    print(find_max_form(["10", "0", "1"], 1, 1))
```



```
lab8/code/dp master x? >
python 3.py
4
2
```

Рисунок 3 – Скриншот работы кода

Сложность алгоритмов

Задача 1.

Условие

Дан размер массива n и число запросов q . Требуется сравнить линейный поиск, бинарный поиск с предсортировкой и хеш-поиск с построением таблицы. Нужно определить, какой подход выгоднее при заданных n и q , учитывая затраты на подготовку.

Решение

Линейный поиск имеет $O(n)$ на запрос без подготовки. Бинарный поиск требует сортировки $O(n \log n)$, после чего каждый запрос $O(\log n)$. Хеш-поиск требует построения $O(n)$, далее $O(1)$ в среднем на запрос. Сопоставляя суммы затрат, выбираем стратегию по q и n .

```
def choose_search_strategy(n, q):
    cost_linear = q * n
    cost_binary = n * (n.bit_length()) + q * (n.bit_length())
    cost_hash = n + q
    if cost_linear <= cost_binary and cost_linear <= cost_hash:
        return "linear"
    if cost_binary <= cost_hash:
        return "binary_search_with_sort"
    return "hash_table"

if __name__ == "__main__":
    print(choose_search_strategy(10**3, 1))
    print(choose_search_strategy(10**5, 100))
    print(choose_search_strategy(10**6, 10**6))
```

```
^ lab8/code/comp  ↵ master  Ⓢ x? >
python 1.py
linear
hash_table
hash_table
```

Рисунок 1 – Скриншот работы кода

Задача 2.

Условие

Дан массив строк `strs` и число `k`. Требуется найти длину максимальной подстроки, которая встречается минимум в `k` различных строках. Сравнить наивный перебор и подход с бинарным поиском по длине и роллинг-хешем.

Решение

Наивный перебор генерирует все подстроки и считает в скольких строках каждая встречается. Оптимизированный подход бинарит по длине `L` и проверяет существование общей подстроки длины `L` через роллинг-хеш множества для каждой строки и пересечение множеств.

```
def longest_common_len_naive(strs, k):
    s0 = strs[0]
    best = 0
    for i in range(len(s0)):
        for j in range(i + 1, len(s0) + 1):
            sub = s0[i:j]
            cnt = 0
            for s in strs:
                if sub in s:
                    cnt += 1
            if cnt >= k and j - i > best:
                best = j - i
    return best

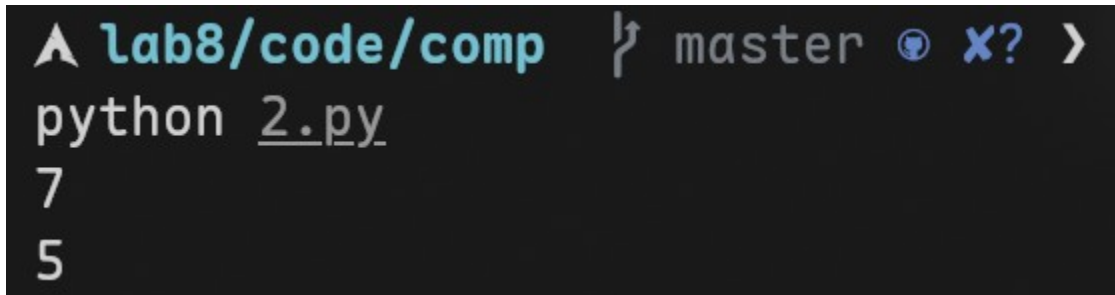
def longest_common_len_optimized(strs, k):
    import sys
    def has_len(L):
        if L == 0:
            return True
        base = 911382323
        mod = 10**9 + 7
```

```

powL = pow(base, L - 1, mod)
commons = None
for s in strs:
    if L > len(s):
        return False
    h = 0
    st = set()
    for t in range(L):
        h = (h * base + ord(s[t])) % mod
    st.add(h)
    for t in range(L, len(s)):
        h = (h - ord(s[t - L]) * powL) % mod
        h = (h * base + ord(s[t])) % mod
        st.add(h)
    if commons is None:
        commons = st
    else:
        commons &= st
    if not commons:
        return False
return len(commons) >= k
lo, hi = 0, min(len(s) for s in strs)
ans = 0
while lo <= hi:
    mid = (lo + hi) // 2
    if has_len(mid):
        ans = mid
        lo = mid + 1
    else:
        hi = mid - 1
return ans

```

```
if __name__ == "__main__":  
    strs = ["abracadabra", "cadabracad", "dabrac"]  
    print(longest_common_len_naive(strs, 2))  
    print(longest_common_len_optimized(strs, 2))
```



```
lab8/code/comp master x? >  
python 2.py  
7  
5
```

Рисунок 2 – Скриншот работы кода

Задача 3.

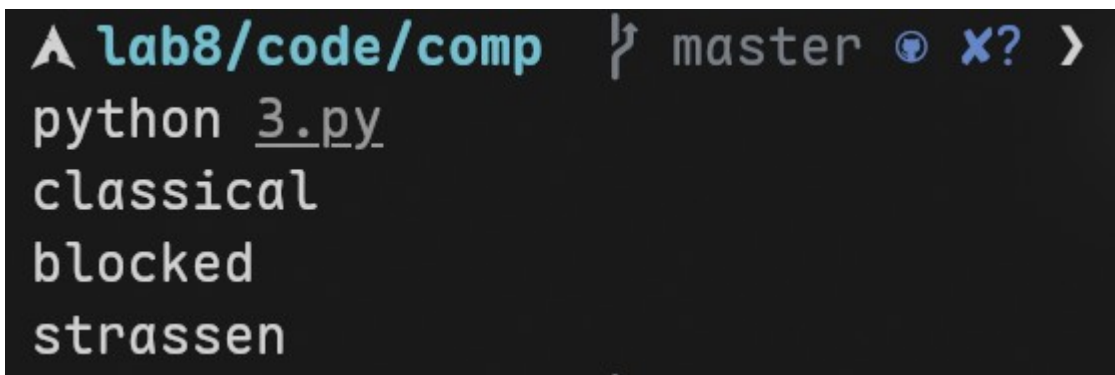
Условие

Для умножения двух матриц $n \times n$ выбрать алгоритм: классический, блочный или Штрассена. Нужно выдать рекомендацию по n с кратким обоснованием.

Решение

Классический $O(n^3)$ прост и устойчив, практичен на малых n . Блочный сохраняет $O(n^3)$, но лучше использует кэш и выигрывает при средних размерах. Штрассена $O(n^{2.807})$ оправдан при больших n при наличии памяти.

```
def choose_mm_algorithm(n):  
    if n <= 64:  
        return "classical"  
    if n <= 512:  
        return "blocked"  
    return "strassen"  
  
if __name__ == "__main__":  
    print(choose_mm_algorithm(64))  
    print(choose_mm_algorithm(256))  
    print(choose_mm_algorithm(1500))
```



```
lab8/code/comp master x? >  
python 3.py  
classical  
blocked  
strassen
```

Рисунок 3 – Скриншот работы кода

Заключение

В ходе лабораторной работы были разработаны задачи по ключевым темам алгоритмов, сформулированы условия, решения и приведены их реализации. Работа позволила закрепить навыки в области алгоритмов и структур данных, а также показала умение анализировать эффективность различных подходов.

Список литературы

1. Aho A.V., Hopcroft J.E., Ullman J.D. Data Structures and Algorithms. – Addison-Wesley, 1983. – 427 p. [Электронный ресурс] – <https://doc.lagout.org/Alfred%20V.%20Aho%20-%20Data%20Structures%20and%20Algorithms.pdf> (12.05.2025).