САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №1

по курсу «Алгоритмы и структуры данных»

Тема: Жадные алгоритмы. Динамическое программирование №2

Вариант 16

Выполнил:

Бархатова Н.А.

К3139

Проверила:

Артамонова В.Е.

Санкт-Петербург

2023 г.

# Содержание отчета

[Содержание отчета 2](#_Toc130507660)

[Задачи по варианту 3](#_Toc130507661)

[Задача №2. Заправки [0.5 балла] 3](#_Toc130507662)

[Задача №3. Максимальный доход от рекламы [0.5 балла] 6](#_Toc130507663)

[Задача №12. Последовательность [1 балл] 8](#_Toc130507664)

[Задача №12. Сувениры [1.5 балла] 11](#_Toc130507665)

[Задача №17. Ход конём [2.5 балла] 13](#_Toc130507666)

[Дополнительные задачи 15](#_Toc130507667)

[Задача №20. Почти палиндром [3 балла] 15](#_Toc130507668)

[Задача №21. Игра в дурака [3 балла] 17](#_Toc130507669)

[Задача №14. Максимальное значение арифметического выражения [2 балла] 20](#_Toc130507670)

[Вывод 23](#_Toc130507671)

# Задачи по варианту

## Задача №2. Заправки [0.5 балла]

Текст задачи:

Вы собираетесь поехать в другой город, расположенный в d км от вашего

родного города. Ваш автомобиль может проехать не более m км на полном баке, и

вы начинаете с полным баком. По пути есть заправочные станции на расстояниях

stop1, stop2, ..., stopn из вашего родного города. Какое минимальное количество

заправок необходимо?

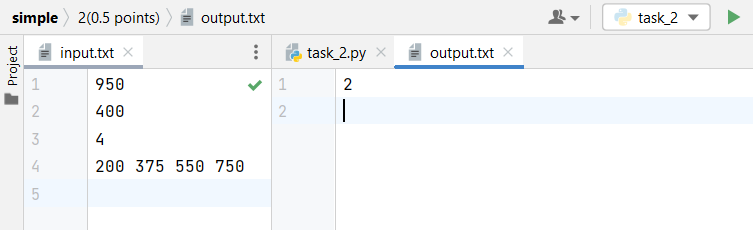
Листинг:

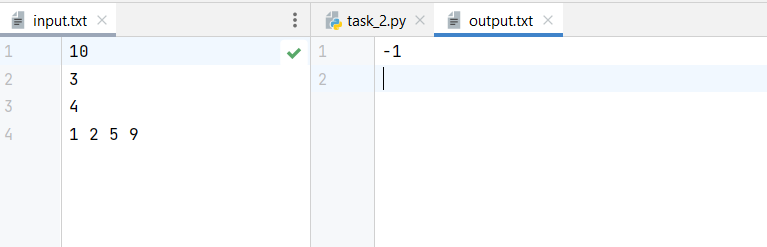
import time  
import tracemalloc  
  
start\_time = time.perf\_counter()  
tracemalloc.start()  
input\_file = open('input.txt')  
d = int(input\_file.readline())  
m = int(input\_file.readline())  
n = int(input\_file.readline())  
stops = list(map(int, input\_file.readline().split()))  
stops.append(d)  
curr\_stop = 0  
fuel = m  
answer = 0  
for i in range(n + 1):  
 if stops[i] - curr\_stop > fuel:  
 if stops[i] - stops[i - 1] > m:  
 answer = -1  
 break  
 curr\_stop = stops[i - 1]  
 fuel = m  
 answer += 1  
 fuel = fuel - (stops[i] - curr\_stop)  
 curr\_stop = stops[i]  
  
with open('output.txt', 'w') as output\_file:  
 print(answer, file=output\_file)  
  
print("Время: ", time.perf\_counter() - start\_time)  
print("Память: ", float(tracemalloc.get\_tracemalloc\_memory()) / (2 \*\* 20))  
tracemalloc.stop()

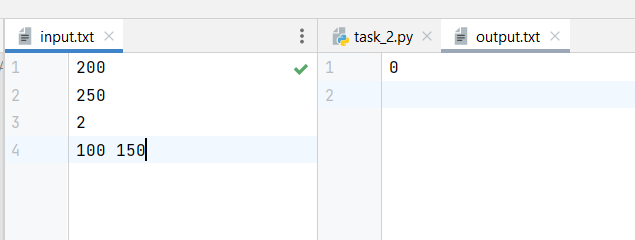
Текстовое объяснение решения:

Считываем входные данные, добавляем в массив stops с заправками длину маршрута (считаем пункт назначения как еще одну остановку). В начале пути бак топлива полный, поэтому fuel = m, начало пути автомобиля на нулевой «координате», следовательно curr\_stop = 0. Проходимся по каждой из заправок (не включая пункт назначения). Если расстояние между заправкой и текущим положением автомобиля меньше, чем количество топлива, то машина едет на заправку (но не заправляется на ней): изменяется количество топлива и текущее положение автомобиля. В какой-то момент количества топлива до следующей заправки N может не хватить. В таком случае автомобилю необходимо было заправиться на предшествующей заправке. Проверяем, теоретически хватит ли автомобилю топлива добраться до N от заправки N-1 при полном баке. Если нет, то автомобиль не сможет доехать до пункта назначения: ответ «-1». Если да, то автомобиль заправляется на заправке N-1, бак полон (fuel = m), отсчет дальнейшего пути идет от позиции заправки N-1, счётчик количества заправок автомобиля увеличивается на 1.

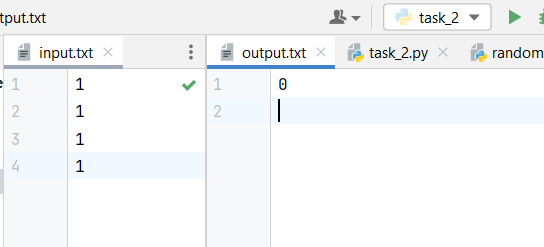
Результат работы кода на примерах из текста задачи:

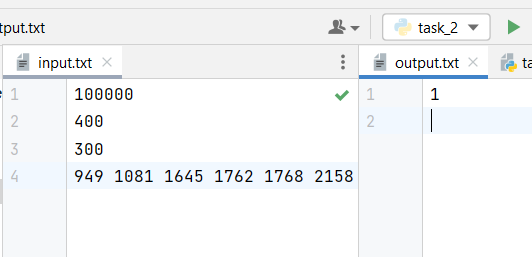






Результат работы кода на максимальных и минимальных значениях:





|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.0007 | 0.0023 |
| Пример из задачи | 0.0007 | 0.0023 |
| Пример из задачи | 0.0007 | 0.0023 |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.00098 | 0.0269 |

Вывод по задаче:

Я научилась решать простейшую задачу на жадный алгоритм.

## Задача №3. Максимальный доход от рекламы [0.5 балла]

Текст задачи:

У вас есть n объявлений для размещения на популярной интернет-странице.

Для каждого объявления вы знаете, сколько рекламодатель готов платить за один клик по этому объявлению. Вы настроили n слотов на своей странице и оценили ожидаемое количество кликов в день для каждого слота. Теперь ваша цель - распределить рекламу по слотам, чтобы максимизировать общий доход.

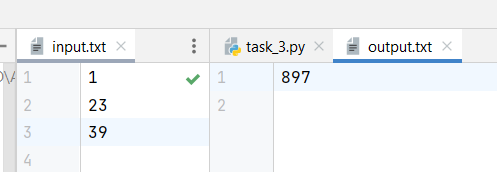
Листинг:

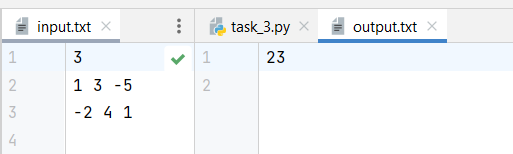
input\_file = open('input.txt')  
n = int(input\_file.readline())  
A = list(map(int, input\_file.readline().split()))  
B = list(map(int, input\_file.readline().split()))  
A.sort(reverse=True)  
B.sort(reverse=True)  
C = [A[i] \* B[i] for i in range(n)]  
with open('output.txt', 'w') as output\_file:  
 print(sum(C), file=output\_file)

Текстовое объяснение решения:

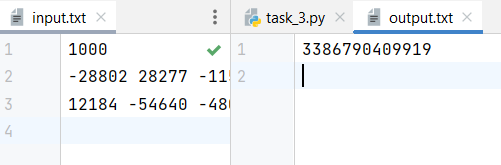
Сортируем по убыванию массив объявлений и массив цен. Умножая максимальную цену на максимальное количество кликов мы получим максимальный доход.

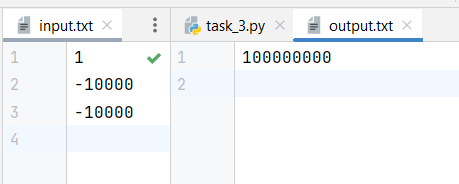
Результат работы кода на примерах из текста задачи:





Результат работы кода на максимальных и минимальных значениях:





|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.0007 | 0.0032 |
| Пример из задачи | 0.0007 | 0.0031 |
| Пример из задачи | 0.0009 | 0.0031 |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.01957 | 0.1560 |

Вывод по задаче:

Я научилась решать еще один тип задач на жадный алгоритм.

## Задача №12. Последовательность [1 балл]

Текст задачи:

Дана последовательность натуральных чисел a1, a2, ..., an, и известно, что ai ≤ i для любого 1 ≤ i ≤ n. Требуется определить, можно ли разбить элементы последовательности на две части таким образом, что сумма элементов в каждой из частей будет равна половине суммы всех элементов последовательности.

1 вариант решения. Использование рекурсии.

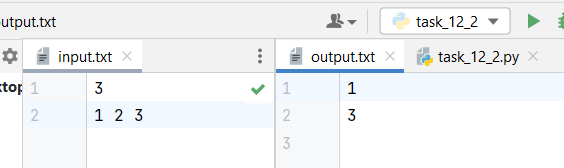
Листинг кода:

import time  
import tracemalloc  
  
start\_time = time.perf\_counter()  
tracemalloc.start()  
  
def isThereASubset(nums, n, len\_of\_subset):  
 N\_index = n - 1  
 if len\_of\_subset == 0:  
 return True  
 if N\_index < 0 or len\_of\_subset < 0:  
 return False  
 include = isThereASubset(nums, N\_index - 1, len\_of\_subset - nums[N\_index])  
 if include:  
 return True  
 exclude = isThereASubset(nums, N\_index - 1, len\_of\_subset)  
 return exclude  
  
  
def counter(S):  
 first\_part = []  
 second\_part = []  
 for i in sorted(S, reverse=True):  
 if sum(S) // 2 >= sum(first\_part) + i:  
 first\_part.append(i)  
 else:  
 second\_part.append(i)  
 return first\_part  
  
  
input\_file = open('input.txt')  
n = int(input\_file.readline())  
nums = list(map(int, input\_file.readline().split()))  
with open('output.txt', 'w') as output\_file:  
 if sum(nums) % 2 == 0:  
 if isThereASubset(nums, n, sum(nums) // 2):  
 print(len(counter(nums)), file=output\_file)  
 print(\*counter(nums), file=output\_file)  
 else:  
 print(-1, file=output\_file)  
  
print("Время: ", time.perf\_counter() - start\_time)  
print("Память: ", float(tracemalloc.get\_tracemalloc\_memory()) / (2 \*\* 20))  
tracemalloc.stop()

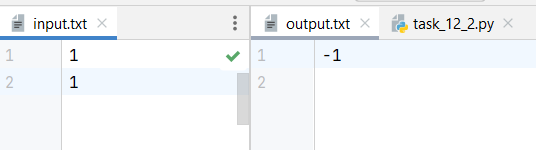
Текстовое объяснение решения:

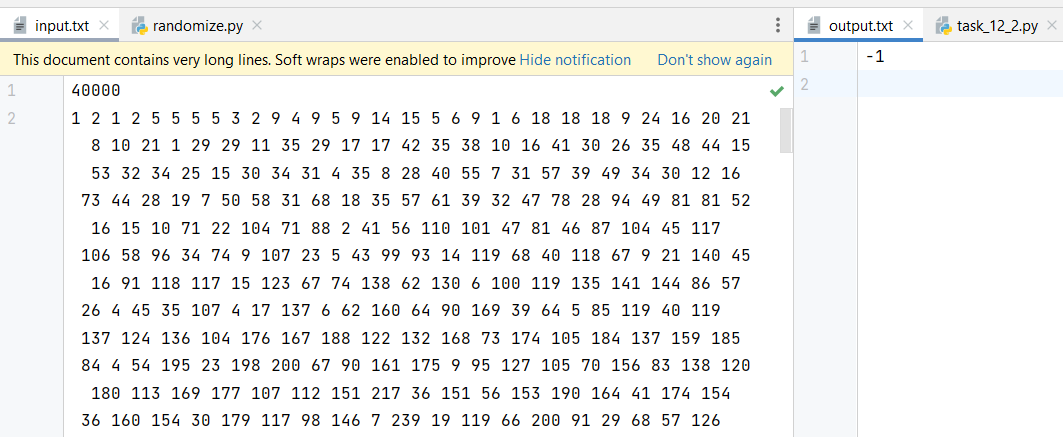
Создадим две функции. Функция isThereASubset определяет, есть ли в множестве подмножество с указанной суммой. Это выясняется с использованием рекурсии. Рассматривая последовательность цифр с конца, мы поочередно пытаемся добавить каждое число. Если это сделать получается, то искомая сумма уменьшается и «курсор» сдвигается дальше. Если нет, то курсор сдвигается без изменения суммы, то есть число пропускается. Функция counter определяет с помощью жадного алгоритма разбиение на 2 части.

Результат работы кода на примерах из текста задачи:



Результат работы кода на максимальных и минимальных значениях:





|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.0008 | 0.0030 |
| Пример из задачи | 0.0012 | 0.0034 |
| Пример из задачи | - | - |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.0563 | 3.1755 |

2 вариант решения. Динамическое программирование.

def isThereASubset(S, n, len\_of\_subset):  
 table = [[False for x in range(len\_of\_subset + 1)] for y in range(n + 1)]  
 for i in range(n + 1):  
 table[i][0] = True  
 for i in range(1, n + 1):  
 for j in range(1, len\_of\_subset + 1):  
 if nums[i - 1] > j:  
 table[i][j] = table[i - 1][j]  
 else:  
 table[i][j] = table[i - 1][j] or table[i - 1][j - nums[i - 1]]  
 return table[n][len\_of\_subset]

Текстовое объяснение решения:

Задачу нахождения подмножества можно свести к задаче о рюкзаке без повторений. Изменим функцию isThereASubset Создадим таблицу с размерами (кол-во цифр в массиве + 1) х (необходимая сумма цифр в подмножестве + 1). Будем заполнять его булевыми значениями. Значение True означает, что массив данной длины содержит подмножество с заданной суммой.

|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.0010 | 0.0030 |
| Пример из задачи | 0.0007 | 0.0034 |
| Пример из задачи | - | - |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.0541 | 3.1747 |

Вывод: Первая функция использует рекурсию и имеет экспоненциальную временную сложность O(2^n). Вторая функция использует динамическое программирование. Она имеет временную сложность O(n\* len\_of\_subset), где n - количество элементов в списке nums, len\_of\_subset - сумма, которую нужно найти в подмножестве.

Таким образом, вторая функция, использующая динамическое программирование, является более эффективной для решения задачи о наличии подмножества, сумма которого равна заданному числу. Она работает быстрее и может обрабатывать большие входные данные, тогда как первая функция сможет обрабатывать только небольшие списки и суммы, из-за своей экспоненциальной сложности.

## Задача №12. Сувениры [1.5 балла]

Текст задачи:

Вы и двое ваших друзей только что вернулись домой после посещения разных стран. Теперь вы хотели бы поровну разделить все сувениры, которые все трое накупили.

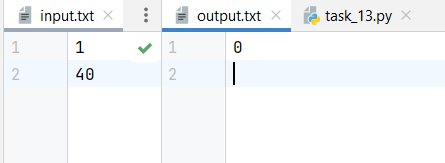
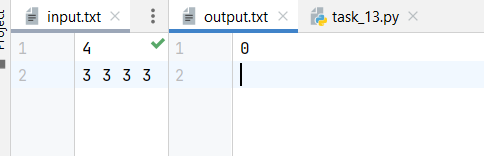
Листинг:

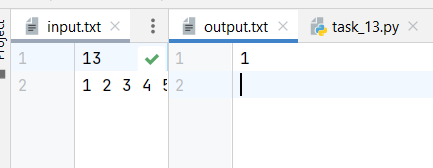
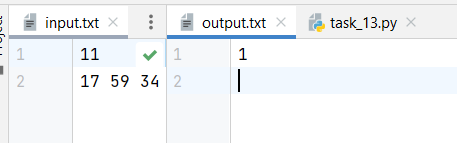
import time  
import tracemalloc  
  
start\_time = time.perf\_counter()  
tracemalloc.start()  
  
input\_file = open('input.txt')  
n = int(input\_file.readline())  
nums = list(map(int, input\_file.readline().split()))  
  
def isThereASubsets(S, n, a, b, c):  
 if a == 0 and b == 0 and c == 0:  
 return True  
 if n < 0:  
 return False  
 part\_a = False  
 if a - S[n] >= 0:  
 part\_a = isThereASubsets(S, n - 1, a - S[n], b, c)  
 part\_b = False  
 if not part\_a and (b - S[n] >= 0):  
 part\_b = isThereASubsets(S, n - 1, a, b - S[n], c)  
 part\_c = False  
 if (not part\_a and not part\_b) and (c - S[n] >= 0):  
 part\_c = isThereASubsets(S, n - 1, a, b, c - S[n])  
 return part\_a or part\_b or part\_c  
  
  
with open('output.txt', 'w') as output\_file:  
 if sum(nums) % 3 == 0 and isThereASubsets(nums, n - 1, sum(nums) // 3, sum(nums) // 3, sum(nums) // 3):  
 print(1, file=output\_file)  
 else:  
 print(0, file=output\_file)  
  
print("Время: ", time.perf\_counter() - start\_time)  
print("Память: ", float(tracemalloc.get\_tracemalloc\_memory()) / (2 \*\* 20))  
tracemalloc.stop()

Текстовое объяснение решения:

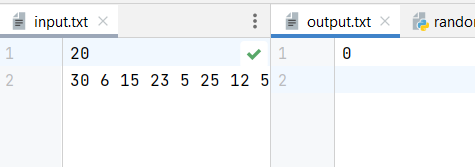
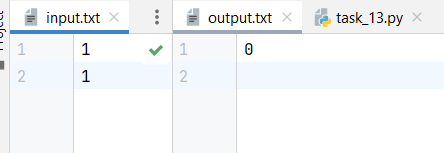
Функция isThereASubsets с помощью рекурсии определяет, возможно ли разделить множество на три подмножества с заданными суммами, поочередно проверяя каждое число и «раскидывая» их по множествам a,b,c

Результат работы кода на примерах из текста задачи:





Результат работы кода на максимальных и минимальных значениях:



|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.0009 | 0.0029 |
| Пример из задачи | 0.0008 | 0.0041 |
| Пример из задачи | 0.0009 | 0.0039 |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.0010 | 0.0029 |

Вывод по задаче: я научилась решать задачу разбиения на 3 произвольных множества

## Задача №17. Ход конём [2.5 балла]

Текст задачи:

Шахматная ассоциация решила оснастить всех своих сотрудников такими телефонными номерами, которые бы набирались на кнопочном телефоне ходом коня. Например, ходом коня набирается телефон 340-49-27. При этом телефонный номер не может начинаться ни с цифры 0, ни с цифры 8.

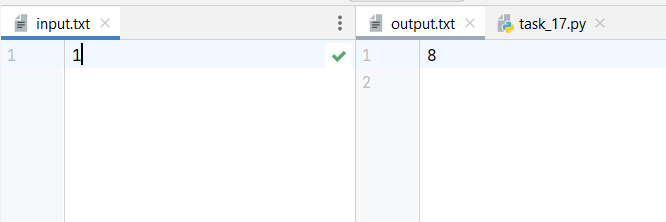
Листинг:

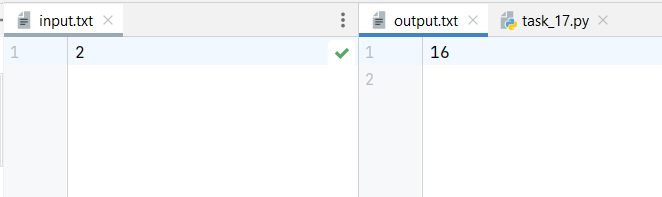
import time  
import tracemalloc  
  
start\_time = time.perf\_counter()  
tracemalloc.start()  
  
input\_file = open('input.txt')  
n = int(input\_file.readline())  
desk = [[4, 6], [6, 8], [7, 9], [4, 8],  
 [0, 3, 9], [], [0, 1, 7], [2, 6], [1, 3], [2, 4]]  
  
M = [[0 for j in range(10)] for i in range(n + 1)]  
  
for j in range(10):  
 if j == 0 or j == 8:  
 continue  
 M[1][j] = 1  
for i in range(n + 1):  
 M[i][0] = 0  
  
for i in range(2, n + 1):  
 for j in range(10):  
 for choosen\_number in desk[j]:  
 M[i][j] += M[i - 1][choosen\_number]  
with open('output.txt', 'w') as output\_file:  
 print(sum(M[n]) % (10 \*\* 9), file=output\_file)  
  
print("Время: ", time.perf\_counter() - start\_time)  
print("Память: ", float(tracemalloc.get\_tracemalloc\_memory()) / (2 \*\* 20))  
tracemalloc.stop()

Текстовое объяснение решения:

Создаем двумерный массив M, размерами 10 х n, где n – длина номера телефона. Заполняем первую строку 0-ми, а вторую единицами (за исключением 0 и 8 столбцов). Для каждой последующей ячейки высчитываем значение как сумму значений из ячеек длины n-1, причем таких, что они могут попасть в число j ходом коня. Ответом является сумма значений ячеек определенной строки n.

Результат работы кода на примерах из текста задачи:





Результат работы кода на максимальных и минимальных значениях:

|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.0007 | 0.0037 |
| Пример из задачи | 0.0007 | 0.0037 |
| Пример из задачи | 0.0007 | 0.0039 |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.0384 | 0.5863 |

Вывод по задаче: я потренировалась в динамическом программировании

# Дополнительные задачи

## Задача №20. Почти палиндром [3 балла]

Текст задачи:

Слово называется палиндромом, если его первая буква совпадает с последней, вторая – с предпоследней и т.д. Например: «abba», «madam», «x». Для заданного числа K слово называется почти палиндромом, если в нем можно изменить не более K любых букв так, чтобы получился палиндром. Например, при K = 2 слова «reactor», «kolobok», «madam» являются почти палиндромами (подчеркнуты буквы, заменой которых можно получить палиндром). Подсловом данного слова являются все слова, получающиеся путем вычеркивания из данного нескольких (возможно, одной или нуля) первых букв и нескольких последних. Например, подсловами слова «cat» являются слова «c», «a», «t», «ca», «at» и само слово «cat» (а «ct» подсловом слова «cat» не является). Требуется для данного числа K определить, сколько подслов данного слова S являются почти палиндромами

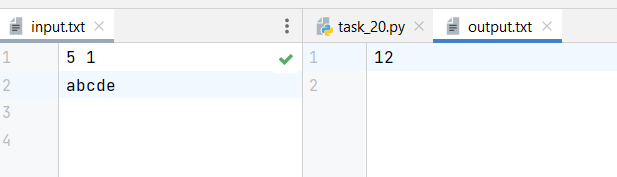
Листинг кода:

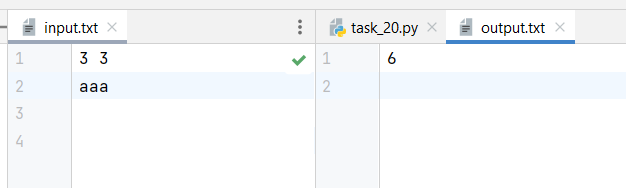
import time  
import tracemalloc  
  
start\_time = time.perf\_counter()  
tracemalloc.start()  
  
input\_file = open('input.txt')  
N, K = map(int, input\_file.readline().split())  
S = input\_file.readline()  
  
M = [[0 if i >= j else 1 for j in range(N)] for i in range(N)]  
for i in range(N):  
 for j in range(N):  
 if i > j:  
 M[i][j] = -1  
 if j - 1 - i == 1:  
 if S[i:j][0] == S[i:j][-1]:  
 M[i][j - 1] = 0  
 else:  
 M[i][j - 1] = 1  
for r in range(2, N):  
 for i in range(N):  
 for j in range(i + 1, N + 1):  
 if j - 1 - i == r:  
 if S[i:j][0] == S[i:j][-1]:  
 M[i][j - 1] = M[i + 1][j - 1 - 1]  
 else:  
 M[i][j - 1] = M[i + 1][j - 1 - 1] + 1  
  
c = 0  
for i in range(N):  
 for j in range(N):  
 if 0 <= M[i][j] <= K:  
 c += 1  
with open('output.txt', 'w') as output\_file:  
 print(c, file=output\_file)  
  
print("Время: ", time.perf\_counter() - start\_time)  
print("Память: ", float(tracemalloc.get\_tracemalloc\_memory()) / (2 \*\* 20))  
tracemalloc.stop()

Текстовое объяснение решения:

Создается массив хранить двумерный массив М где М[i][j] будет содержать количество замен, необходимых для превращения подстроки от индекса i до индекса j в палиндром. Значения на диагонали массива будут равны 0, так как однобуквенная подстрока всегда является палиндромом. Заполняет массив по диагоналям. Вторая диагональ состоит из подстрок длины 2. Третья и следующие стоки формируются на основе первых двух.

Результат работы кода на примерах из текста задачи:





|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи |  |  |
| Пример из задачи | 0.0008 | 0.0037 |
| Пример из задачи | 0.0007 | 0.0036 |
| Верхняя граница диапазона значений входных данных из текста задачи |  |  |

Вывод по задаче: В данном решении есть ошибка, так как код не проходит тесты на acmp. В целом задача очень интересная и полезная. Научилась заполнять массив по диагонали

## Задача №21. Игра в дурака [3 балла]

Текст задачи:

Петя очень любит программировать. Недавно он решил реализовать популярную карточную игру «Дурак». Но у Пети пока маловато опыта, ему срочно нужна Ваша помощь. Как известно, в «Дурака» играют колодой из 36 карт. В Петиной программе каждая карта представляется в виде строки из двух символов, где первый символ означает ранг (‘6’, ‘7’, ‘8’, ‘9’, ‘T’, ‘J’, ‘Q’, ‘K’, ‘A’) карты, а второй символ означает масть (‘S’, ‘C’, ‘D’, ‘H’). Ранги перечислены в порядке возрастания старшинства. Пете необходимо решить следующую задачу: сможет ли игрок, обладая набором из N карт, отбить M карт, которыми под него сделан ход? Для того чтобы отбиться, игроку нужно покрыть каждую из карт, которыми под него сделан ход, картой из своей колоды. Карту можно покрыть либо старшей картой той же масти, либо картой козырной масти. Если кроющаяся карта сама является козырной, то её можно покрыть только старшим козырем. Одной картой можно покрыть только одну карту.

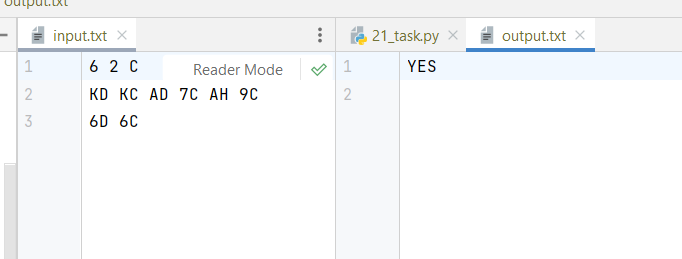
Листинг кода:

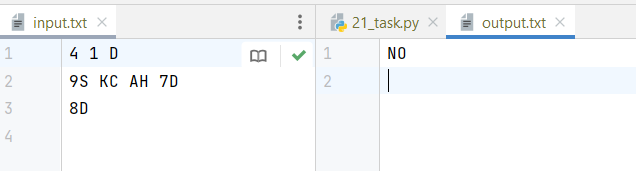
import time  
import tracemalloc  
  
start\_time = time.perf\_counter()  
tracemalloc.start()  
  
input\_file = open('input.txt')  
N, M, R = input\_file.readline().split()  
  
  
def sort\_key(rang):  
 return RANGS.index(rang[0])  
  
answer = 'YES'  
RANGS = ['6', '7', '8', '9', 'T', 'J', 'Q', 'K', 'A']  
my\_cards = sorted(input\_file.readline().split(), key=sort\_key)  
enemy\_cards = sorted(input\_file.readline().split(), key=sort\_key)  
my\_cards\_dict = {}  
for card in my\_cards:  
 rang = card[0]  
 color = card[1]  
 if color in my\_cards\_dict.keys():  
 my\_cards\_dict[color].append(rang)  
 else:  
 my\_cards\_dict[color] = [rang]  
for enemy\_card in enemy\_cards:  
 isCovered = False  
 enemy\_rang = enemy\_card[0]  
 enemy\_color = enemy\_card[1]  
 if enemy\_color in my\_cards\_dict.keys():  
 for rang in my\_cards\_dict[enemy\_color]:  
 if RANGS.index(rang) > RANGS.index(enemy\_rang):  
 my\_cards\_dict[enemy\_color].remove(rang)  
 isCovered = True  
  
 if enemy\_color != R and not (isCovered) and R in my\_cards\_dict.keys() and len(my\_cards\_dict[R]) > 0:  
 my\_cards\_dict[R].remove(my\_cards\_dict[R][0])  
 isCovered = True  
  
 if not (isCovered):  
 answer = "NO"  
 break  
with open('output.txt', 'w') as output\_file:  
 print(answer, file=output\_file)  
print("Время: ", time.perf\_counter() - start\_time)  
print("Память: ", float(tracemalloc.get\_tracemalloc\_memory()) / (2 \*\* 20))  
tracemalloc.stop()

Текстовое объяснение решения:

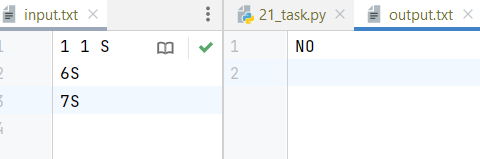
Сохраняю входные данные в списки (список своих карт и список карт противника) и сортирую их по рангу. Создаю словарь, где ключ – масть, значение – ранг. Проходя по каждой из вражеских карт, я определяю, можно ли её покрыть обычным способом (старшей картой той же масти), если нет, то пытаюсь покрыть козырной картой.

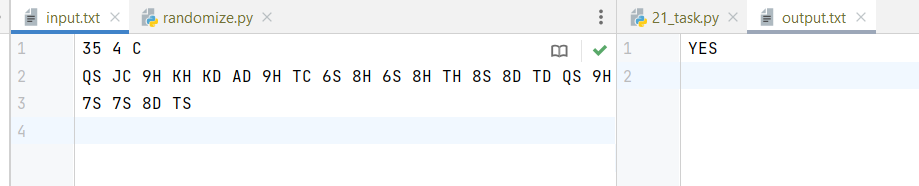
Результат работы кода на примерах из текста задачи:



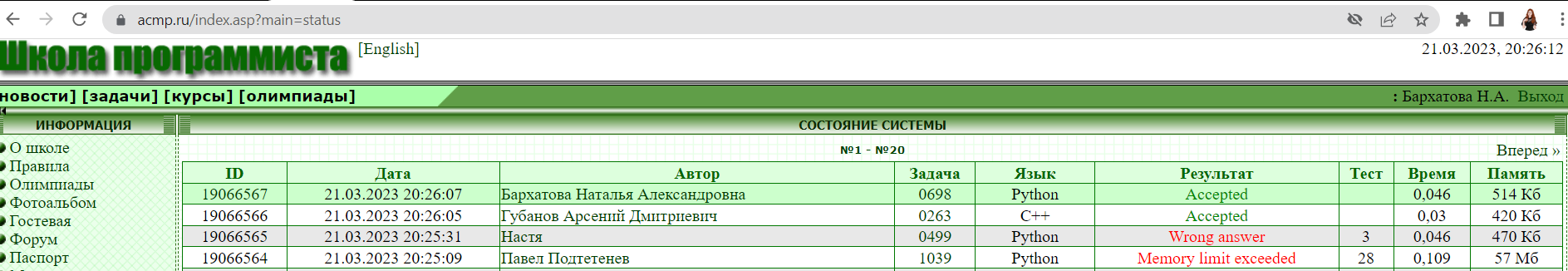


Результат работы кода на максимальных и минимальных значениях:





Проверка задачи на acmp:



|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.0008 | 0.0037 |
| Пример из задачи | 0.0008 | 0.0037 |
| Пример из задачи | 0.0007 | 0.0036 |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.0007 | 0.0059 |

Вывод по задаче: не самая интересная задача, решается перебором вариантов.

## Задача №14. Максимальное значение арифметического выражения [2 балла]

Текст задачи:

В этой задаче ваша цель - добавить скобки к заданному арифметическому

выражению, чтобы максимизировать его значение.

max(5 − 8 + 7 × 4 − 8 + 9) =?

Найдите максимальное значение арифметического выражения, указав порядок применения его арифметических операций с по-

мощью дополнительных скобок.

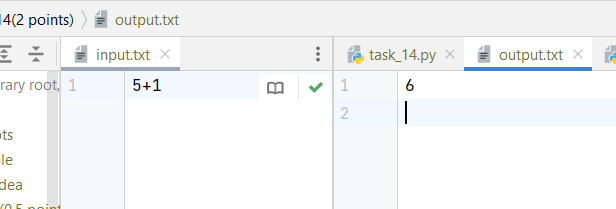
Листинг кода:

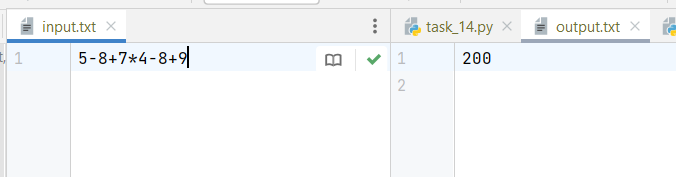
import re  
import time  
import tracemalloc  
  
start\_time = time.perf\_counter()  
tracemalloc.start()  
  
input\_file = open('input.txt')  
s = input\_file.readline()  
numbers = [i for i in re.split('|\+|-|\\*', s) if i]  
signs = [i for i in re.split('\d', s) if i]  
  
  
def minimal\_and\_maximum(i, j, max\_values, min\_values, signs):  
 minimal = float("+inf")  
 maximum = float("-inf")  
 for k in range(i, j):  
 if signs[k] == "+":  
 a, b, c, d = max\_values[i][k] + max\_values[k + 1][j], max\_values[i][k] + min\_values[k + 1][j], min\_values[i][k] + max\_values[k + 1][j], min\_values[i][k] + min\_values[k + 1][j]  
 elif signs[k] == "\*":  
 a, b, c, d = max\_values[i][k] \* max\_values[k + 1][j], max\_values[i][k] \* min\_values[k + 1][j], min\_values[i][k] \* max\_values[k + 1][j], min\_values[i][k] \* min\_values[k + 1][j]  
 else:  
 a, b, c, d = max\_values[i][k] - max\_values[k + 1][j], max\_values[i][k] - min\_values[k + 1][j], min\_values[i][k] - max\_values[k + 1][j], min\_values[i][k] - min\_values[k + 1][j]  
 minimal = min(minimal, a, b, c, d)  
 maximum = max(maximum, a, b, c, d)  
 return maximum, minimal  
  
  
N = len(numbers)  
min\_values, max\_values = [[0 for i in range(N)] for j in range(N)], [[0 for i in range(N)] for j in range(N)]  
for i in range(N):  
 max\_values[i][i], min\_values[i][i] = int(s[i \* 2]), int(s[i \* 2])  
for diff in range(1, N):  
 for i in range(N - diff):  
 j = i + diff  
 max\_values[i][j], min\_values[i][j] = minimal\_and\_maximum(i, j, max\_values, min\_values, signs)  
with open('output.txt', 'w') as output\_file:  
 print(max\_values[0][N - 1], file=output\_file)  
  
print("Время: ", time.perf\_counter() - start\_time)  
print("Память: ", float(tracemalloc.get\_tracemalloc\_memory()) / (2 \*\* 20))  
tracemalloc.stop()

Текстовое объяснение решения:

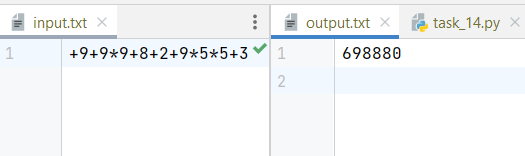
Есть два массива: массив максимальных и массив минимальных значений фрагментов выражения. Индексы элементов этого массива — это начало и конец фрагмента выражения. Перебрав все возможные коэффициенты (j всегда больше i) определяем минимальное и максимальное значение для данного фрагмента. Каждый последующий фрагмент высчитывается на основе полученных до этого данных. Ответ в правом верхнем углу массива.

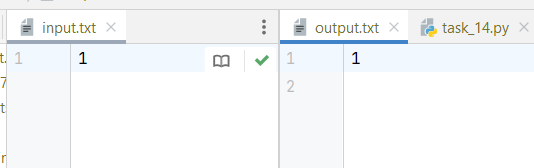
Результат работы кода на примерах из текста задачи:





Результат работы кода на максимальных и минимальных значениях:





|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.0014 | 0.0082 |
| Пример из задачи | 0.0012 | 0.0085 |
| Пример из задачи | 0.0015 | 0.0101 |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.0038 | 0.0159 |

Вывод по задаче: изначально я очень хотела решить её другим способом, но в итоге у меня не получилось. Грустно. Есть подозрение, что данное в описании решение очень плохо сказывается на памяти

# Вывод

Жадные алгоритмы и динамическое программирование являются двумя фундаментальными подходами к решению задач оптимизации. Оба подхода могут быть использованы для нахождения оптимальных решений, но имеют разные преимущества и ограничения.

Жадные алгоритмы часто применяются в случаях, когда необходимо принимать локальные оптимальные решения на каждом шаге, чтобы достичь глобальной оптимальности.

Динамическое программирование широко используется для решения задач, которые могут быть разбиты на меньшие подзадачи. Этот подход позволяет сохранять результаты решения каждой подзадачи, чтобы избежать повторных вычислений, и строит оптимальное решение путем комбинации решений подзадач.

Эти два подхода представляют мощные инструменты для оптимизации, и понимание их принципов может помочь при решении широкого спектра задач.