# 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

Выполнила:

Олейник П.Д.

K3143

Проверил:

Афанасьев А.В.

Санкт-Петербург 2024 г.

# Оглавление

Задачи	3
Задача №3. Максимальный доход от рекламы	3
Задача №6. Максимальная зарплата	6
Задача №11. Максимальное количество золота	10
Задача №14. Максимальное значение арифметического выражения	13
Задача №16. Продавец	16
Reiron	10

#### Задачи

#### Задача №3. Максимальный доход от рекламы

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.

- Постановка задачи. Даны две последовательности  $a_1, a_2, ..., a_n$  ( $a_i$  прибыль за клик по i-му объявлению) и  $b_1, b_2, ..., b_n$  ( $b_i$  среднее количество кликов в день i-го слота), нужно разбить их на n пар ( $a_i, b_j$ ) так, чтобы сумма их произведений была максимальной.
- Формат ввода / входного файла (input.txt). В первой строке содержится целое число n, во второй последовательность целых чисел  $a_1, a_2, ..., a_n$ , в третьей последовательность целых чисел  $b_1, b_2, ..., b_n$ .
- Ограничения на входные данные.  $1 \le n \le 10^3, -10^5 \le a_i, b_i \le 10^5,$  для всех  $1 \le i \le n.$
- Формат вывода / выходного файла (output.txt). Выведите максимальное значение  $\sum_{i=1}^{n} a_i c_i$ , где  $c_1, c_2, ..., c_n$  является перестановкой  $b_1, b_2, ..., b_n$ .
- Ограничение по времени. 2 сек.
- Примеры:

input.txt	output.txt	input.txt	output.txt
1	897	3	23
23		1 3 -5	
39		-2 4 1	

Во втором примере  $23 = 3 \cdot 4 + 1 \cdot 1 + (-5) \cdot (-2)$ .

```
import time
import tracemalloc

t_start = time.perf_counter()
tracemalloc.start()
infinity = float("inf")

def merge(array, p, q, r):
    n1 = q - p + 1
    n2 = r - q
    array1 = array[p: q + 1] + [infinity]
    array2 = array[q + 1: r + 1] + [infinity]
    i, j = 0, 0
    for k in range(p, r + 1):
```

```
if array1[i] <= array2[j]:</pre>
            array[k] = array1[i]
        else:
            array[k] = array2[j]
def merge_sort(array, p, r):
   if p < r:
       q = (p + r) // 2
       merge_sort(array, p, q)
       merge_sort(array, q + 1, r)
       merge(array, p, q, r)
       return array
    return array
def solve(A, B, n):
    sorted_A = [i for i in merge_sort(A, 0, n - 1)]
    sorted_B = [i for i in merge_sort(B, 0, n - 1)]
    return sum([sorted_A[i] * sorted_B[i] for i in range(n)])
def write_to_file():
   n = int(f.readline())
   A = [int(i) for i in f.readline().split()]
    B = [int(i) for i in f.readline().split()]
   f.close()
    result = solve(A, B, n)
   file2 = open("output3.txt", "w+")
    file2.write(str(result))
if __name__ == "__main__":
   write_to_file()
    print("Время выполнения: %s секунд " % (time.perf_counter() - t_start))
    print("Затраты памяти: %s КБ " % (tracemalloc.get_traced_memory()[0] / 2
** 10))
```

Жадный алгоритм заключается в том, чтобы на каждом шаге среди свободных слотов выбирать слот с наибольшим количеством кликов и размещать в нем наиболее дорогое объявление из еще неразмещенных. Для этого нужно отсортировать оба массива и найти произведения і-ых элементов. Сортировку массивов можно оценить в O(nlogn) (вслучае сортировки слиянием,

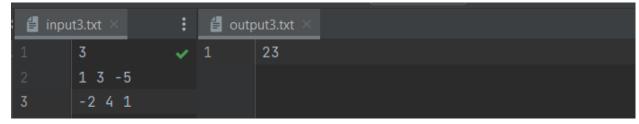
## например)

Результат работы кода на примерах из текста задачи:

inpu	it3.txt ×	:	a out	put3.txt ×
1	1	~	1	897
2	23			
3	39			

Время выполнения: 0.007969300000695512 секунд

Затраты памяти: 4.7109375 KБ



Время выполнения: 0.005485500005306676 секунд

Затраты памяти: 5.2578125 KБ

Результат работы кода на максимальных и минимальных значениях:



Время выполнения: 0.02759430000151042 секунд

Затраты памяти: 8.244140625 КБ



Время выполнения: 0.005792199997813441 секунд

Затраты памяти: 4.7109375 КБ

Вывод по задаче: данная задача имеет оптимальное решение.

## Задача №6. Максимальная зарплата

В качестве последнего вопроса успешного собеседования ваш начальник дает вам несколько листков бумаги с цифрами и просит составить из этих цифр наибольшее число. Полученное число будет вашей зарплатой, поэтому вы очень заинтересованы в максимизации этого числа. Как вы можете это сделать?

На лекциях мы рассмотрели следующий алгоритм составления наибольшего числа из заданных *однозначных* чисел.

```
def LargestNumber(Digits):
    answer = ''
    while Digits:
        maxDigit = float('-inf')
        for digit in Digits:
            if digit >= maxDigit:
                 maxDigit = digit
            answer += str(maxDigit)
            Digits.remove(maxDigit)
        return answer
```

К сожалению, этот алгоритм работает только в том случае, если вход состоит из однозначных чисел. Например, для ввода, состоящего из двух целых чисел 23 и 3 (23 не однозначное число!) возвращается 233, в то время как наибольшее число на самом деле равно 323. Другими словами, использование наибольшего числа из входных данных в качестве первого числа не является безопасным ходом.

Ваша цель в этой задаче – настроить описанный выше алгоритм так, чтобы он работал не только с однозначными числами, но и с произвольными положительными целыми числами.

- Постановка задачи. Составить наибольшее число из набора целых чисел.
- Формат ввода / входного файла (input.txt). Первая строка входных данных содержит целое число n. Во второй строке даны целые числа  $a_1, a_2, ..., a_n$ .
- Ограничения на входные данные.  $1 \le n \le 10^2, \, 1 \le a_i \le 10^3$  для всех  $1 \le i \le n$ .
- Формат вывода / выходного файла (output.txt). Выведите наибольшее число, которое можно составить из  $a_1, a_2, ..., a_n$ .
- Ограничение по времени. 2 сек.

	input.txt	output.txt	input.txt	output.txt
	2	221	3	923923
l	21 2		23 39 92	

```
import time
import tracemalloc

t_start = time.perf_counter()
tracemalloc.start()
```

```
def LargestNumber(digits):
   answer = ""
   while digits:
       maxdigit = digits[0]
       for digit in digits:
           if first_is_better(digit, maxdigit):
               maxdigit = digit
        answer += maxdigit
        digits.remove(maxdigit)
   return answer
def first_is_better(a, b):
   if a[0] > b[0]:
       return True
   if a[0] < b[0]:
       return False
   if len(a) == len(b):
       return a > b
   if len(a) > 1 and len(b) > 1:
       return first_is_better(a[1:], b[1:])
   if len(a) == 1:
       return a > b[1]
   if len(b) == 1:
       return a[1] > b
def solve(digits):
   return LargestNumber(digits)
def write_to_file():
   f = open("input6.txt")
   n = int(f.readline())
   digits = [i for i in f.readline().split()]
   f.close()
   result = solve(digits)
   file2 = open("output6.txt", "w+")
   file2.write(result)
if __name__ == "__main__":
   write_to_file()
   print("Время выполнения: %s секунд " % (time.perf_counter() - t_start))
   print("Затраты памяти: %s КБ " % (tracemalloc.get_traced_memory()[0] / 2
** 10))
```

Для того, чтобы алгоритм работал для n-значных чисел, необходимо ввести новое правило сравнения чисел. Оно реализовано в функции first\_is\_better.

Наилучшее из двух чисел — то, у которого первая цифра больше. Если первые цифры одинаковы и длины чисел одинаковы, то наилучшее то, которое больше по правилу сравнения строк. Если одно из чисел короче, то нужно отбросить все совпадающие цифры в начале и прийти либо к одному из описанных выше случаев, либо к случаю, когда длина одного из двух новых чисел равна 1 и сравнить это число и вторую цифру второго числа.

## Результат работы кода на примерах из текста задачи:

<b>inpu</b>	t6.txt ×	:	a outp	out6.txt ×
1	3	~	1	923923
2	23 39 92			

Время выполнения: 0.007372299995040521 секунд

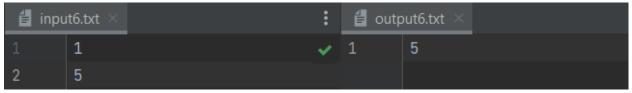
Затраты памяти: 3.421875 КБ

🎒 inpu	ıt6.txt ×	:	a outp	out6.txt ×
1	2	~	1	221
2	21 2			

Время выполнения: 0.0073921000002883375 секунд

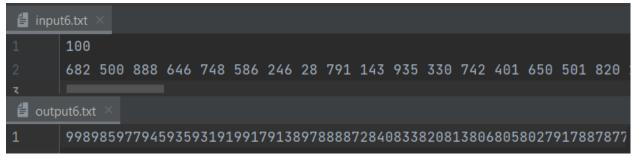
Затраты памяти: 3.421875 КБ

## Результат работы кода на максимальных и минимальных значениях:



Время выполнения: 0.0008209000079659745 секунд

Затраты памяти: 3.421875 КБ



Время выполнения: 0.002152599990949966 секунд

Затраты памяти: 3.7216796875 КБ

Вывод: данная задача решается с помощью жадного алгоритма, приведенный код дает возможность составлять наибольшее число не только из цифр, но и из чисел.

## Залача №11. Максимальное количество золота

Вам дается набор золотых слитков, и ваша цель - набрать как можно больше золота в свою сумку. Существует только одна копия каждого слитка, и для каждого слитка вы можете либо взять его, либо нет (т.е. вы не можете взять часть слитка).

- Постановка задачи. Даны n золотых слитков, найдите максимальный вес золота, который поместится в сумку вместимостью W.
- Формат ввода / входного файла (input.txt). Первая строка входных данных содержит вместимость W сумки и количество n золотых слитков. В следующей строке записано n целых чисел  $w_0, w_1, ..., w_{n-1}$ , определяющие вес золотых слитков.
- Ограничения на входные данные.  $1 \le W \le 10^4, \ 1 \le n \le 300, \ 0 \le w_0, ..., w_{n-1} \le 10^5$
- Формат вывода / выходного файла (output.txt). Выведите максимальный вес золота, который поместится в сумку вместимости W.
- Ограничение по времени. 5 сек.
- Пример:

input.txt	output.txt
10 3	9
1 4 8	

Здесь сумма весов первого и последнего слитка равна 9.

• Обратите внимание, что в этой задаче все предметы имеют одинаковую стоимость на единицу веса по простой причине: все они сделаны из золота.

```
import time
import tracemalloc

t_start = time.perf_counter()
tracemalloc.start()

def solve(W, n, weights):
    dp = [[0] * (W+1) for _ in range(n+1)]
    for j in range(1, n + 1): # j - сколько первых элементов weights можно
брать
    for i in range(1, W+1): # i - вместимость рюкзака
        dp[j][i] = dp[j-1][i]
        if i >= weights[j - 1]:
              dp[j][i] = max(dp[j][i], dp[j-1][i - weights[j - 1]] +
weights[j - 1])
    return dp[n][W]

def write_to_file():
```

```
f = open("input6.txt")
W, n = [int(i) for i in f.readline().split()]
w = [int(i) for i in f.readline().split()]
f.close()

result = solve(W, n, w)
file2 = open("output6.txt", "w+")
file2.write(str(result))

if __name__ == "__main__":
    write_to_file()

    print("Время выполнения: %s секунд " % (time.perf_counter() - t_start))
    print("Затраты памяти: %s КБ " % (tracemalloc.get_traced_memory()[0] / 2
** 10))
```

Вся логика реализована в функции solve. Создаем таблицу dp (в каждой строке W+1 ячейка, всего n+1 строка). Dp[j][i] — максимальный вес для рюкзака вместимостью і при условии, что брать можно только первые ј элементов из списков элементов weights. Проходим таблицу построчно слева направо и для каждой ячейки вычисляем максимальное значение как максимум из двух случаев (когда j-ый элемент берется или не берется). Сложность O(nW)

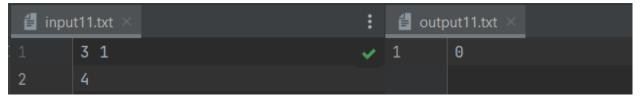
Результат работы кода на примерах из текста задачи:

🛔 input	11.txt ×	:	autput11.txt ×	
1	10 3	~	1	9
2	1 4 8			

Время выполнения: 0.007331899993005209 секунд

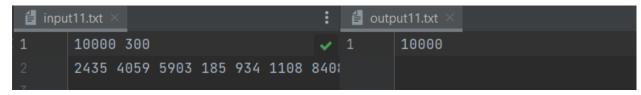
Затраты памяти: 3.21484375 КБ

Результат работы кода на максимальных и минимальных значениях:



Время выполнения: 0.00724489999932386 секунд

Затраты памяти: 3.21484375 КБ



Время выполнения: 4.988885400001891 секунд

Затраты памяти: 7.53515625 КБ

Вывод: данная задача решается динамическим подходом с помощью создания двумерной таблицы.

## Задача №14. Максимальное значение арифметического выражения

В этой задаче ваша цель - добавить скобки к заданному арифметическому выражению, чтобы максимизировать его значение.

$$max(5-8+7\times 4-8+9)=?$$

- Постановка задачи. Найдите максимальное значение арифметического выражения, указав порядок применения его арифметических операций с помощью дополнительных скобок.
- Формат ввода / входного файла (input.txt). Единственная строка входных данных содержит строку s длины 2n+1 для некоторого n с символами  $s_0, s_1, ..., s_{2n}$ . Каждый символ в четной позиции s является цифрой (то есть целым числом от 0 до 9), а каждый символ в нечетной позиции является одной из трех операций из +,-,\*
- Ограничения на входные данные.  $0 \le n \le 14$  (следовательно, строка содержит не более 29 символов).
- Формат вывода / выходного файла (output.txt). Выведите максимально возможное значение заданного арифметического выражения среди различных порядков применения арифметических операций.
- Ограничение по времени. 5 сек.
- Пример:

input.txt	output.txt	input.txt	output.txt
1+5	6	5-8+7*4-8+9	200

```
Здесь 200 = (5 - ((8+7)*(4-(8+9)))).
```

```
import time
import tracemalloc
t_start = time.perf_counter()
tracemalloc.start()
def calc(a, b, s):
    if s == "+":
    if s == "-":
       return a - b
    if s == "*":
        return a * b
def solve(s):
    nums = [int(s[i]) for i in range(0, len(s), 2)]
    ops = [s[i] for i in range(1, len(s), 2)]
    n = len(nums)
    m = [[0] * n for _ in range(n)]
    M = [[0] * n for _ in range(n)]
    for i in range(n):
        m[i][i] = nums[i]
        M[i][i] = nums[i]
    for s in range(1, n):
```

```
for i in range(n-s):
            j = i + s
            m[i][j], M[i][j] = min_and_max(i, j, m, M, ops)
    return M[0][n-1]
def min_and_max(i, j, m, M, ops):
    maximum = float("inf") * (-1)
   minimum = float("inf")
    for k in range(i, j):
        a = calc(M[i][k], M[k+1][j], ops[k])
        b = calc(m[i][k], M[k+1][j], ops[k])
        c = calc(M[i][k], m[k+1][j], ops[k])
        d = calc(m[i][k], m[k+1][j], ops[k])
        maximum = max(maximum, a, b, c, d)
        minimum = min(minimum, a, b, c, d)
    return minimum, maximum
def write_to_file():
    f = open("input14.txt")
   s = f.readline()
   f.close()
    result = solve(s)
   file2 = open("output14.txt", "w+")
    file2.write(str(result))
if __name__ == "__main__":
   write_to_file()
    print("Время выполнения: %s секунд " % (time.perf_counter() - t_start))
    print("Затраты памяти: %s КБ " % (tracemalloc.get_traced_memory()[0] / 2
** 10))
```

Создаем две двумерные таблицу n\*n, где n-kon-во чисел в строке. M[i][j]- максимальное число которое можно получить из среза строки [i,j], m[i][j]- минимальное число которое можно получить из среза строки [i,j]Обходить будем верхнюю часть таблиц над главной диагональю. Порядок обхода — по диагоналям сверху вниз в сторону самой верхней и самой правой ячейки. На каждой итерации алгоритма функция  $min\_and\_max$  вычисляет наименьшее и наибольшее значения для среза строки [i,j], рассматривая всевозможные комбинации наименьших и наибольших результатов для подстрок строки [i,j]. Сложность алгоритма  $O(n^3)$  [вместо перебора, где O(n!)].

Результат работы кода на примерах из текста задачи:



Время выполнения: 0.0016082000074675307 секунд

Затраты памяти: 4.83984375 КБ

<b>inpu</b>	t14.txt ×	:	a outp	out14.txt ×	
1	5-8+7*4-8+9	~	1	200	

Время выполнения: 0.00749540000106208 секунд

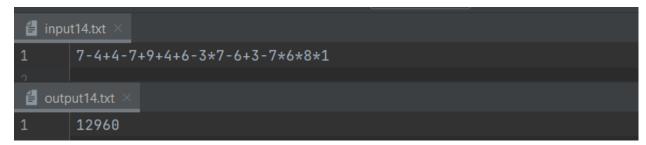
Затраты памяти: 4.83984375 КБ

Результат работы кода на максимальных и минимальных значениях:

🛔 inpu	t14.txt ×	:	al outp	out14.txt ×	
1	2	~	1	2	

Время выполнения: 0.00168679999478627 секунд

Затраты памяти: 3.91796875 КБ



Время выполнения: 0.009361500007798895 секунд

Затраты памяти: 5.60546875 КБ

#### Вывод:

Данная задача решается с помощью динамического подхода и создания двумерной таблицы. Отличие данной задачи от задачи 11 состоит в порядке обхода таблицы и соответственно в способе вычисления значения для очередной ячейки таблицы.

## Задача №16. Продавец

- Постановка задачи. Продавец техники хочет объехать n городов, посетив каждый из них ровно один раз. Помогите ему найти кратчайший путь.
- Формат ввода / входного файла (input.txt). Первая строка входного файла содержит натуральное число n количество городов. Следующие n строк содержат по n чисел длины путей между городами. В i-й строке j-е число  $a_{i,j}$  это расстояние между городами i и j.
- Ограничения на входные данные.  $1 \le n \le 13, 0 \le a_{i,j} \le 10^6, a_{i,j} = a_{j,i}, a_{i,i} = 0.$
- Формат вывода / выходного файла (output.txt). В первой строке выходного файла выведите длину кратчайшего пути. Во второй строке выведите пчисел порядок, в котором нужно посетить города.
- Ограничение по времени. 1 сек.
- Ограничение по памяти. 256 мб.
- Пример:

input.txt
5
0 183 163 173 181
183 0 165 172 171
163 165 0 189 302
173 172 189 0 167
181 171 302 167 0
output.txt
666
4 5 2 3 1

```
import time
import tracemalloc

t_start = time.perf_counter()
tracemalloc.start()

def solve(dist, n):
    infinity = float("inf")
    min_length = infinity
    path = []
    # перебираем все варианты, в какой вершине может начинаться гамильтонов

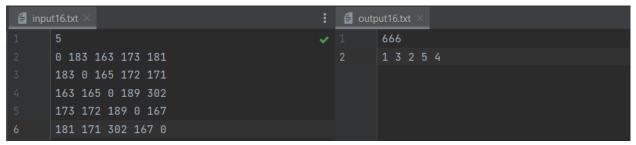
путь
    for start in range(n):
        visited = [start]
        length = 0

        for _ in range(n - 1):
            curr_city = visited[-1]
```

```
min_dist = float('inf')
            close_city = -1
            for city in range(n):
                if city not in visited and dist[curr_city][city] < min_dist:</pre>
                    min_dist = dist[curr_city][city]
                    close_city = city
            visited.append(close_city)
           length += min_dist
        if length < min_length:</pre>
           min_length = length
            path = visited
   return min_length, [x+1 for x in path]
def write_to_file():
   f = open("input16.txt")
   n = int(f.readline())
   dist = [[int(el) for el in f.readline().split()] for _ in range(n)]
   f.close()
   lenght, path = solve(dist, n)
   file2 = open("output16.txt", "w+")
   file2.write(str(lenght) + "\n")
   file2.write(f'{" ".join(str(x) for x in path)}')
   file2.close()
if __name__ == "__main__":
   write_to_file()
   print("Время выполнения: %s секунд " % (time.perf_counter() - t_start))
   print("Затраты памяти: %s КБ " % (tracemalloc.get_traced_memory()[0] / 2
** 10))
```

Для каждой вершины найдем гамильтонов путь минимальной длины и среди всех таких путей найдем наименьший по длине. Чтобы найти гамильтонов путь, начинающийся в вершине start воспользуемся жадным подходом. Будем хранить список вершин, которые мы уже обошли и на каждом шаге выбирать вершину, которая ближе к последней посещенной. Считается, что граф полный, те из каждой вершины можно попасть в каждую за один шаг.

Результат работы кода на примерах из текста задачи:



Время выполнения: 0.0008056000078795478 секунд

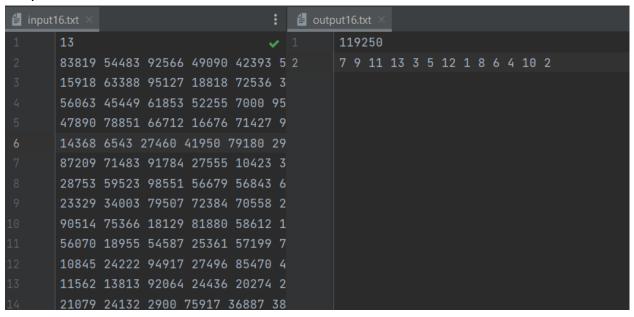
Затраты памяти: 3.3857421875 КБ

Результат работы кода на максимальных и минимальных значениях:

input1	:16.txt ×		a outp	out16.txt ×
1 :	1	~		0
2 !	5		2	1

Время выполнения: 0.011978299997281283 секунд

Затраты памяти: 3.3857421875 КБ



Время выполнения: 0.0015505999908782542 секунд

Затраты памяти: 3.3857421875 КБ

#### Вывод:

Данная задача решается с помощью жадного алгоритма (при условии, что граф полный).

#### Вывод

Жадные алгоритмы и динамическое программирование одни из наиболее простых методов оптимизации. При использовании жадной стратегии необходимо быть внимательным и проверять, является ли совершаемый ход безопасным ходом. В задачах, где жадная стратегия не гарантирует безопасного хода, имеет смысл рассмотреть динамический подход. Чаще всего, чтобы решить задачу с помощью динамического подхода, достаточно посмотреть на решение с конца или(и) разбить задачу на более маленькие подзадачи.