

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.
К3143

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
Вывод.....	19

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, \dots, a_n (a_i - прибыль за клик по i -му объявлению) и b_1, b_2, \dots, 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, \dots, a_n , в третьей - последовательность целых чисел b_1, b_2, \dots, b_n .
- **Ограничения на входные данные.** $1 \leq n \leq 10^3$, $-10^5 \leq a_i, b_i \leq 10^5$, для всех $1 \leq i \leq n$.
- **Формат вывода / выходного файла (output.txt).** Выведите максимальное значение $\sum_{i=1}^n a_i c_i$, где c_1, c_2, \dots, c_n является перестановкой b_1, b_2, \dots, 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 $23 = 3 \cdot 4 + 1 \cdot 1 + (-5) \cdot (-2)$.

Листинг кода:

```
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]:
            array[k] = array1[i]
            i += 1
        else:
            array[k] = array2[j]
            j += 1

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

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():
    f = open("input3.txt")
    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))

```

Текстовое объяснение решения.

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 i -ых элементов. Сортировку массивов можно оценить в $O(n \log n)$ (вслучае сортировки слиянием,

например)

Результат работы кода на примерах из текста задачи:

input3.txt			output3.txt	
1	1	✓	1	897
2	23			
3	39			

Время выполнения: 0.007969300000695512 секунд

Затраты памяти: 4.7109375 КБ

input3.txt			output3.txt	
1	3	✓	1	23
2	1 3 -5			
3	-2 4 1			

Время выполнения: 0.005485500005306676 секунд

Затраты памяти: 5.2578125 КБ

Результат работы кода на максимальных и минимальных значениях:

input3.txt			output3.txt	
1	1000	✓	1	34301990915
2	-5434 -44 -9931 -26			
3	-9776 2372 -4732 -4			
4				

Время выполнения: 0.02759430000151042 секунд

Затраты памяти: 8.244140625 КБ

inp			output3.txt	
1	1	✓	1	6
2	2			
3	3			

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 даст вам несколько листов 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.

```
1 def LargestNumber(Digits):
2     answer = ''
3     while Digits:
4         maxDigit = float('-inf')
5         for digit in Digits:
6             if digit >= maxDigit:
7                 maxDigit = digit
8         answer += str(maxDigit)
9         Digits.remove(maxDigit)
10    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, \dots, a_n .
- **Ограничения на входные данные.** $1 \leq n \leq 10^2$, $1 \leq a_i \leq 10^3$ для всех $1 \leq i \leq n$.
- **Формат вывода / выходного файла (output.txt).** Выведите наибольшее число, которое можно составить из a_1, a_2, \dots, a_n .
- Ограничение по времени. 2 сек.

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

Листинг кода:

```
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))

```

Текстовое объяснение решения.

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:

input6.txt			output6.txt	
1	3	✓	1	923923
2	23 39 92			

Время выполнения: 0.007372299995040521 секунд

Затраты памяти: 3.421875 КБ

input6.txt			output6.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:

input6.txt			output6.txt	
1	1	✓	1	5
2	5			

Время выполнения: 0.0008209000079659745 секунд

Затраты памяти: 3.421875 КБ

input6.txt			output6.txt	
1	100			
2	682 500 888 646 748 586 246 28 791 143 935 330 742 401 650 501 820			
3				
output6.txt				
1	9989859779459359319199179138978888728408338208138068058027917887877			

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.

Задача №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, \dots, w_{n-1} , определяющие вес золотых слитков.
- **Ограничения на входные данные.** $1 \leq W \leq 10^4$, $1 \leq n \leq 300$, $0 \leq w_0, \dots, w_{n-1} \leq 10^5$
- **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.

Листинг кода:

```
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))

```

Текстовое объяснение решения.

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 i при условии, что брать можно только первые j 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:

input11.txt			output11.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:

input11.txt			output11.txt	
1	3 1	✓	1	0
2	4			

Время выполнения: 0.007244899999932386 секунд

Затраты памяти: 3.21484375 КБ

input11.txt						output11.txt	
1	10000	300	✓	1		10000	
2	2435	4059	5903	185	934	1108	8401
3							

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, \dots, 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.** $0 \leq n \leq 14$ (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 $200 = (5 - ((8 + 7) * (4 - (8 + 9))))$.

Листинг кода:

```
import time
import tracemalloc

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

def calc(a, b, s):
    if s == "+":
        return a + b
    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))

```

Текстовое объяснение решения.

Создаем две двумерные таблицы $n \times n$, где n – кол-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:

input14.txt ×		output14.txt ×	
1	1+5	✓	1 6

Время выполнения: 0.0016082000074675307 секунд

Затраты памяти: 4.83984375 КБ

input14.txt ×		output14.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:

input14.txt ×		output14.txt ×	
1	2	✓	1 2

Время выполнения: 0.00168679999478627 секунд

Затраты памяти: 3.91796875 КБ

input14.txt ×		output14.txt ×	
1	7-4+4-7+9+4+6-3*7-6+3-7*6*8*1		
2			
1	12960		

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.** $1 \leq n \leq 13$, $0 \leq a_{i,j} \leq 10^6$, $a_{i,j} = a_{j,i}$, $a_{i,i} = 0$.
- **Формат вывода / выходного файла (output.txt).** В первой строке выходного файла выведите длину кратчайшего пути. Во второй строке выведите н чисел – 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

Листинг кода:

```
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:
                min_dist = dist[curr_city][city]
                close_city = city
        visited.append(close_city)
        length += min_dist
    if length < min_length:
        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(e1) for e1 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))

```

Текстовое объяснение решения.

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, т.е. из каждой вершины можно попасть в каждую за один шаг.

Результат работы кода на примерах из текста задачи:

input16.txt ×					output16.txt ×				
1	5				1	666			
2	0 183 163 173 181				2	1 3 2 5 4			
3	183 0 165 172 171								
4	163 165 0 189 302								
5	173 172 189 0 167								
6	181 171 302 167 0								

Время выполнения: 0.0008056000078795478 секунд

Затраты памяти: 3.3857421875 КБ

Результат работы кода на максимальных и минимальных значениях:

input16.txt ×					output16.txt ×				
1	1				1	0			
2	5				2	1			

Время выполнения: 0.011978299997281283 секунд

Затраты памяти: 3.3857421875 КБ

input16.txt ×					output16.txt ×				
1	13				1	119250			
2	83819 54483 92566 49090 42393 5 2					7 9 11 13 3 5 12 1 8 6 4 10 2			
3	15918 63388 95127 18818 72536 3								
4	56063 45449 61853 52255 7000 95								
5	47890 78851 66712 16676 71427 9								
6	14368 6543 27460 41950 79180 29								
7	87209 71483 91784 27555 10423 3								
8	28753 59523 98551 56679 56843 6								
9	23329 34003 79507 72384 70558 2								
10	90514 75366 18129 81880 58612 1								
11	56070 18955 54587 25361 57199 7								
12	10845 24222 94917 27496 85470 4								
13	11562 13813 92064 24436 20274 2								
14	21079 24132 2900 75917 36887 38								

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.