

**Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University**

Отчет по лабораторной работе № 2-1

По дисциплине Алгоритмы и структуры данных

Обучающийся Овсянкин Даниил Витальевич

Преподаватель: Ромакина О.М

Факультет Инфокоммуникационных технологий

Группа К3244

Направление подготовки 45.03.04 Интеллектуальные системы в гуманитарной сфере

Образовательная программа Интеллектуальные системы в гуманитарной сфере

Санкт-Петербург

2025 г.

1 задача. Максимальная стоимость добычи (0.5 балла)

Вор находит гораздо больше добычи, чем может поместиться в его сумке. Помогите ему найти самую ценную комбинацию предметов, предполагая, что любая часть предмета добычи может быть помещена в его сумку.

Цель - реализовать алгоритм для задачи о дробном рюкзаке.

```
def solve(): 1 usage
    with open('input.txt', 'r', encoding='utf-8') as fin:
        n, W = map(int, fin.readline().split())
        items = []
        zero_weight_value = 0.0
        for _ in range(n):
            p, w = map(int, fin.readline().split())
            if w == 0:
                if p > 0:
                    zero_weight_value += float(p)
            else:
                items.append((p, w, p / w))
        items.sort(key=lambda x: x[2], reverse=True)

        total_value = zero_weight_value
        capacity = float(W)
        for p, w, d in items:
            if capacity <= 0:
                break
            take = min(capacity, float(w))
            total_value += d * take
            capacity -= take

        with open('output.txt', 'w', encoding='utf-8') as fout:
            fout.write(f"{total_value:.4f}")

if __name__ == "__main__":
    solve()
```

Текстовое объяснение решения:

Программа читает количество предметов и вместимость сумки, затем пары «цена-вес». Предметы с ненулевым весом сортируются по убыванию отношения цены к весу. После сортировки алгоритм последовательно добавляет в сумку каждый предмет: если он полностью помещается, берётся целиком, иначе берётся дробная часть, равная оставшейся вместимости, с добавлением к суммарной ценности произведения удельной стоимости на взятый вес. Предметы нулевого веса, если их ценность положительна, учитываются сразу, так как не занимают место. Итоговая ценность выводится с точностью до четырёх знаков после запятой.

Результат на примере:

Пример №1

```
h.py  input.txt x
3 50
60 20
100 50
120 30
180.0000
```

Пример №2

```
h.py  input.txt x
1 10
500 30
166.6667
```

Вывод:

Задача решается жадно по плотности ценности; сложность $O(n \log n)$, точность вывода удовлетворяет требованию 10^{-3} .

3 задача. Максимальный доход от рекламы (0.5 балла)

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.

```
def read_n_ints(fin, n): 2 usages
    vals = []
    while len(vals) < n:
        line = fin.readline()
        if not line:
            break
        vals.extend(map(int, line.split()))
    if len(vals) != n:
        raise ValueError("Ожидалось %d чисел, получено %d" % (n, len(vals)))
    return vals

def solve(): 1 usage
    with open('input.txt', 'r', encoding='utf-8') as fin:
        first = fin.readline()
        while first is not None and first.strip() == '':
            first = fin.readline()
        n = int(first.strip())

        a = read_n_ints(fin, n)
        b = read_n_ints(fin, n)
```

```

    a.sort()
    b.sort()
    ans = sum(x * y for x, y in zip(a, b))

    with open('output.txt', 'w', encoding='utf-8') as fout:
        fout.write(str(ans))

if __name__ == "__main__":
    solve()
```

Текстовое объяснение решения:

Программа читает количество объявлений и две последовательности: цену за клик для каждого объявления и ожидаемые клики по слотам. Затем обе последовательности сортируются в одном направлении, после чего сумма произведений соответствующих элементов вычисляется и выводится. Оптимальность обеспечивается неравенством о перестановках: попарное соединение отсортированных массивов даёт наибольшую возможную сумму. Время работы $O(n \log n)$ из-за сортировки, память $O(1)$ сверх входа.

Результат на примере:

Пример №1

```
n.py  input.txt x
1
23
39 | 897 |
```

Пример №2

```
n.py  input.txt x
3
1 3
-5
-2 4 1 | 23 |
```

Вывод:

Максимальный доход от рекламы достигается попарным умножением одинаково отсортированных массивов.

8 задача. Расписание лекций (1 балл)

- **Постановка задачи.** Вы наверно знаете, что в ИТМО лекции читают одни из лучших преподаватели мира. К сожалению, лекционных аудиторий у нас не так уж и много, особенно на Биржевой, поэтому каждый преподаватель составил список лекций, которые он хочет прочитать студентам. Чтобы студенты, в начале февраля, увидели расписание лекций, необходимо его составить прямо сейчас. И без вас нам здесь не справиться. У нас есть список заявок от преподавателей на лекции для одной из аудиторий. Каждая заявка представлена в виде временного интервала $[s_i, f_i)$ - время начала и конца лекции. Лекция считается открытым интервалом, то есть какая-то лекция может начаться в момент окончания другой, без перерыва. Необходимо выбрать из этих заявок такое подмножество, чтобы суммарно выполнить максимальное количество заявок. Учтите, что одновременно в лекционной аудитории, конечно же, может читаться лишь одна лекция.

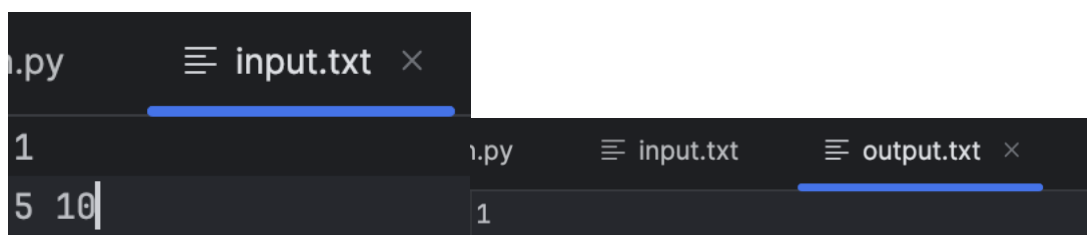
```
def solve(): 1 usage
    with open('input.txt', 'r', encoding='utf-8') as fin:
        line = fin.readline()
        while line and line.strip() == '':
            line = fin.readline()
        n = int(line.strip())
        intervals = []
        for _ in range(n):
            line = fin.readline()
            while line and line.strip() == '':
                line = fin.readline()
            s, f = map(int, line.split())
            intervals.append((f, s))
        intervals.sort()
        count = 0
        last_end = -1
        for f, s in intervals:
            if s >= last_end:
                count += 1
                last_end = f
        with open('output.txt', 'w', encoding='utf-8') as fout:
            fout.write(str(count))
```

Текстовое объяснение решения:

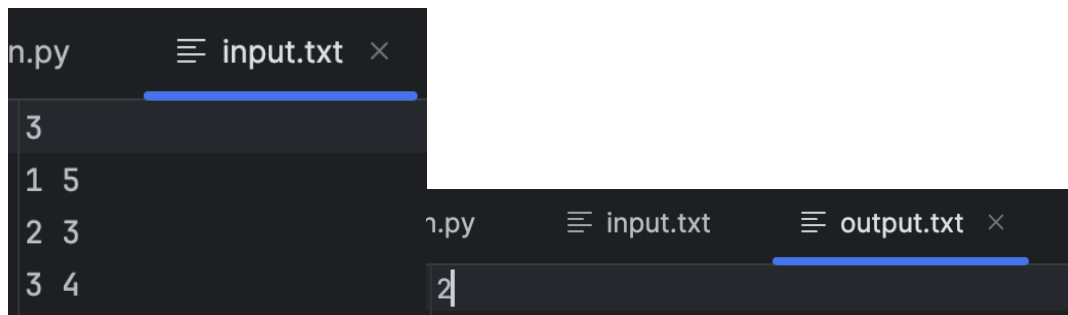
В задаче было реализовано жадное планирование: интервалы заявок сортируются по времени окончания, после чего выбираются по порядку только те лекции, чьё время начала не меньше конца последней уже выбранной лекции, так как интервалы открытые и стык по границе разрешён. Такой выбор максимизирует число совместимых заявок; время работы $O(N \log N)$, дополнительная память $O(1)$ сверх входа.

Результат на примере:

Пример №1



Пример №2



```
n.py  input.txt x
3
1 5
2 3
3 4

n.py  input.txt  output.txt x
2
```

Вывод:

В задаче было реализовано жадное планирование — сортировка заявок по времени окончания и последовательный выбор непересекающихся интервалов, начиная с самых ранних по финишу. Такой подход гарантирует максимум допустимых лекций при сложности $O(N \log N)$.

14 задача. Максимальное значение арифметического выражения (2 балла)

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.

```

def apply(op: str, a: int, b: int) -> int: 4 usages
    if op == '+':
        return a + b
    if op == '-':
        return a - b
    return a * b

def solve(): 1 usage
    with open('input.txt', 'r', encoding='utf-8') as fin:
        s = fin.read().strip().replace(_old: ' ', _new: '')

    if not s:
        ans = 0
    elif len(s) == 1:
        ans = int(s)
    else:
        nums = [int(s[i]) for i in range(0, len(s), 2)]
        ops = [s[i] for i in range(1, len(s), 2)]
        m = len(nums)

```

```

INF = 10**18
dp_min = [[0]*m for _ in range(m)]
dp_max = [[0]*m for _ in range(m)]
for i in range(m):
    dp_min[i][i] = dp_max[i][i] = nums[i]

for L in range(2, m+1):
    for i in range(0, m-L+1):
        j = i + L - 1
        mn, mx = INF, -INF
        for k in range(i, j):
            op = ops[k]
            a1, a2 = dp_min[i][k], dp_max[i][k]
            b1, b2 = dp_min[k+1][j], dp_max[k+1][j]
            candidates = (
                apply(op, a1, b1),
                apply(op, a1, b2),
                apply(op, a2, b1),
                apply(op, a2, b2),
            )

```

```

            mn = min(mn, *candidates)
            mx = max(mx, *candidates)
            dp_min[i][j] = mn
            dp_max[i][j] = mx

    ans = dp_max[0][m-1]

    with open('output.txt', 'w', encoding='utf-8') as fout:
        fout.write(str(ans))

if __name__ == "__main__":
    solve()

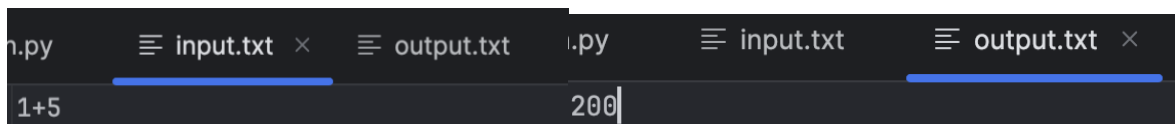
```

Текстовое объяснение решения:

В задаче было реализовано динамическое программирование по подотрезкам: выражение разбивается на числа и операции, для каждого подотрезка поддерживаются минимально и максимально достижимые значения, а при расширении подотрезка перебираются все позиции разреза и комбинируются значения слева и справа с учётом текущей операции. Это гарантирует нахождение глобального максимума, поскольку рассматриваются все корректные порядки расстановки скобок. Временная сложность $O(n^3)$, память $O(n^2)$.

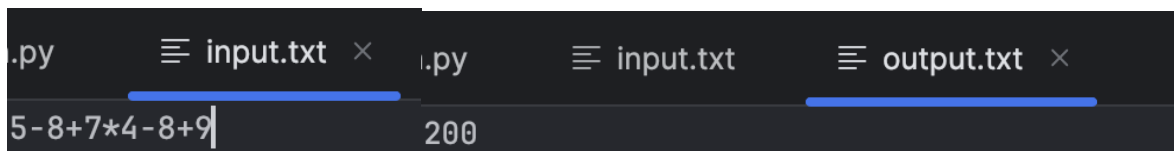
Результат на примере:

Пример №1



```
1.py  input.txt x  output.txt  1.py  input.txt  output.txt x
1+5  200
```

Пример №2



```
1.py  input.txt x  output.txt  1.py  input.txt  output.txt x
5-8+7*4-8+9  200
```

Вывод: задача решалась методом DP min-max на подотрезках; перебор всех разрезов и учёт минимумов и максимумов обеспечивают максимум выражения среди всех расстановок скобок.

17 задача. Ход конем (2.5 балла)

- **Постановка 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.

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |
| . | 0 | . |

Напишите программу, определяющую количество телефонных номеров длины N , набираемых ходом коня. Поскольку таких номеров может быть очень много, выведите ответ по модулю 10^9 .

```

MOD = 10 ** 9

MOVES = {
    0: [4, 6],
    1: [6, 8],
    2: [7, 9],
    3: [4, 8],
    4: [0, 3, 9],
    5: [],
    6: [0, 1, 7],
    7: [2, 6],
    8: [1, 3],
    9: [2, 4],
}

def solve(): 1 usage
    with open('input.txt', 'r', encoding='utf-8') as fin:
        n_line = fin.read().strip()
        N = int(n_line)

    # База длина 1 — нельзя начинать с 0 и 8
    dp = [0] * 10
    for d in [1, 2, 3, 4, 5, 6, 7, 9]:
        dp[d] = 1

    for _ in range(2, N + 1):
        nxt = [0] * 10
        for d in range(10):
            if dp[d] == 0:
                continue
            for t in MOVES[d]:
                nxt[t] = (nxt[t] + dp[d]) % MOD
        dp = nxt

    ans = sum(dp) % MOD if N >= 1 else 0

    with open('output.txt', 'w', encoding='utf-8') as fout:
        fout.write(str(ans))

if __name__ == "__main__":
    solve()

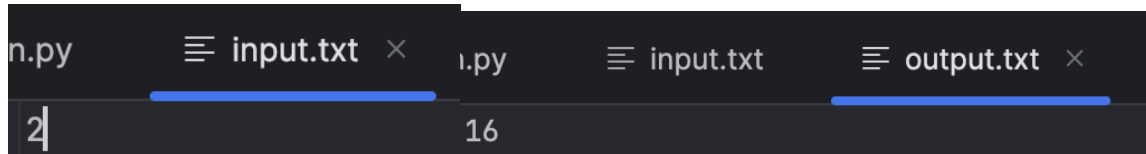
```

Текстовое объяснение решения:

В задаче было реализовано динамическое программирование по длине и последней цифре: базовый слой учитывает запрет на старт с 0 и 8, затем для каждой длины пересчитываются количества по разрешённым конёвым переходам между цифрами клавиатуры, после чего результат берётся как

сумма количеств по всем конечным цифрам с вычислением по модулю 10^9 .
Время работы $O(10 \cdot N)$, память $O(10)$.

Результат на примере:



The screenshot shows a code editor interface. At the top, there are tabs for 'n.py', 'input.txt', and 'output.txt'. The 'n.py' tab is active, showing a single line of code with the number '2'. The 'input.txt' tab is also visible, showing the number '16'. The 'output.txt' tab is visible but empty. The editor has a dark theme with blue highlights for the input and output values.

Вывод: задача решалась ДП на графе ходов коня — итеративное обновление вектора из 10 состояний дало искомое число номеров по модулю 10^9

20 задача. Почти палиндром (3 балла)

- **Постановка 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 «с», «а», «t», «са», «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.

```
def count_k_almost_palindromes(s: str, k: int) -> int: 1 usage
    n = len(s)
    total = 0

    # нечётные длины
    for c in range(n):
        l = r = c
        mism = 0
        while l >= 0 and r < n:
            if s[l] != s[r]:
                mism += 1
            if mism > k:
                break
            total += 1
            l -= 1
            r += 1

    # чётные длины
    for c in range(n - 1):
        l, r = c, c + 1
        mism = 0
        while l >= 0 and r < n:
            if s[l] != s[r]:
```

```
                mism += 1
            if mism > k:
                break
            total += 1
            l -= 1
            r += 1

    return total

def solve(): 1 usage
    with open('input.txt', 'r', encoding='utf-8') as fin:
        n_k = fin.readline().split()
        n, k = int(n_k[0]), int(n_k[1])
        s = fin.readline().strip()

    ans = count_k_almost_palindromes(s, k)

    with open('output.txt', 'w', encoding='utf-8') as fout:
        fout.write(str(ans))
```

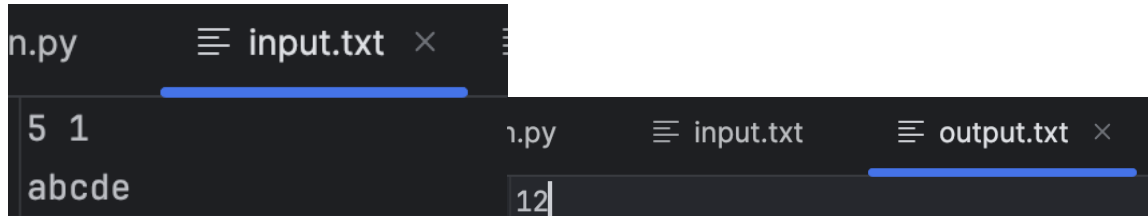
Текстовое объяснение решения:

В задаче было реализовано расширение от каждого центра подстроки как при поиске палиндромов: для каждого центра (нечётного и чётного) границы двигаются симметрично, а счётчик фиксирует число пар символов, которые не совпали; пока это число не превосходит заданное K , текущая подстрока засчитывается. Такой подсчёт перебирает все возможные подстроки и

определяет, можно ли сделать их палиндромом, изменив не более K символов.

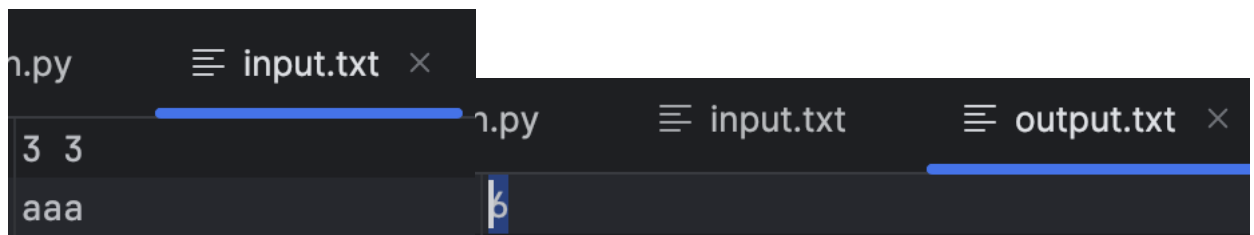
Результат на примере:

Пример №1



```
n.py  input.txt  x
5 1
abcde
12
n.py  input.txt  output.txt  x
```

Пример №2



```
n.py  input.txt  x
3 3
aaa
6
n.py  input.txt  output.txt  x
```

Вывод: задача решалась двухпроходным расширением от центров с учётом числа несовпадений; итоговая сложность $O(N^2)$, память $O(1)$.

19 задача. Произведение матриц (3 балла)

- **Постановка задачи.** В произведении последовательности матриц полностью расставлены скобки, если выполняется один из следующих пунктов:
 - Произведение состоит из одной матрицы.
 - Оно является заключённым в скобки произведением двух произведений с полностью расставленными скобками.

Полная расстановка скобок называется оптимальной, если количество операций, требуемых для вычисления произведения, минимально.

Требуется найти оптимальную расстановку скобок в произведении последовательности матриц.

```

def build_parentheses(s, i, j): 3 usages
    if i == j:
        return "A"
    k = s[i][j]
    return "(" + build_parentheses(s, i, k) + build_parentheses(s, k + 1, j) + ")"

def solve(): 1 usage
    with open('input.txt', 'r', encoding='utf-8') as fin:
        n_line = fin.readline()
        while n_line and n_line.strip() == '':
            n_line = fin.readline()
        n = int(n_line)

        p = []
        if n >= 1:
            a, b = map(int, fin.readline().split())
            p = [a, b]
            for _ in range(1, n):
                a, b = map(int, fin.readline().split())
                p.append(b)

```

```

    if n == 1:
        ans = "A"
    else:
        # DP-таблицы m — минимальная стоимость, s — место разреза
        m = [[0] * n for _ in range(n)]
        s = [[0] * n for _ in range(n)]

        # L — длина цепочки
        for L in range(2, n + 1):
            for i in range(0, n - L + 1):
                j = i + L - 1
                best = 10**18
                best_k = i
                # перебор разреза
                for k in range(i, j):
                    cost = (m[i][k] +
                           m[k + 1][j] +
                           p[i] * p[k + 1] * p[j + 1])
                    if cost < best:
                        best = cost
                        best_k = k

```

```

                m[i][j] = best
                s[i][j] = best_k

        ans = build_parentheses(s, i: 0, n - 1)

        with open('output.txt', 'w', encoding='utf-8') as fout:
            fout.write(ans)

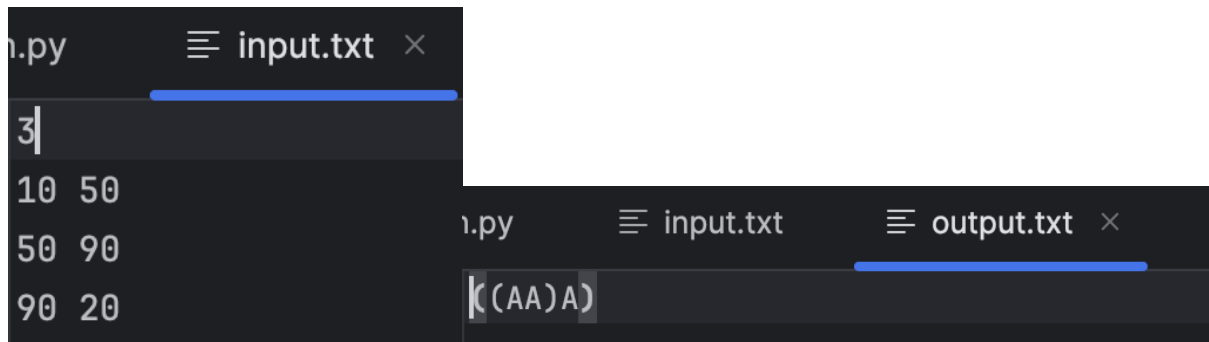
if __name__ == "__main__":
    solve()

```

Текстовое объяснение решения:

В задаче было реализовано динамическое программирование по подцепочкам матриц: для каждой пары границ i, j вычисляется минимальная стоимость перемножения $A_i \dots A_j$ с перебором всех позиций разреза k и формулой стоимости $m[i, k] + m[k+1, j] + r_i r_{k+1} r_j +$, где r - последовательность совместимых размеров. Параллельно сохраняется оптимальный разрез, после чего скобочная запись восстанавливается рекурсивно, представляя каждую матрицу символом A . Временная сложность $O(n^3)$, память $O(n^2)$.

Результат на примере:



```
input.txt
3
10 50
50 90
90 20

output.txt
K(AA)A
```

Вывод: задача решалась ДП для цепочки матриц, после подсчёта минимальных стоимостей по всем отрезкам восстановлена оптимальная скобочная структура и выведена строка вида $((AA)A)$.

Вывод по лабораторной работе

Практиковался в решении задач и лучше ознакомился с темами про жадные алгоритмы и динамическое программирование