Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Алгоритмы и структуры данных

Лабораторная работа №2_1

Жадные алгоритмы.

Динамическое программирование №2

Выполнил:

Бараканов Жаргал Мырзабекович

Факультет ИКТ

Группа К3121

Преподаватель:

Харьковская Татьяна Александровна

Санкт-Петербург

2022

Задание 1.

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.

- **Формат ввода / входного файла (input.txt).** В первой строке входных данных задано целое число n - количество предметов, и W - вместимость сумки. Следующие n строк определяют значения веса и стоимости предметов. В i-ой строке содержатся целые числа p_i и w_i – стоимость и вес i-го предмета, соответственно.
- **Ограничения на входные данные.** $1 \le n \le 10^3$, $0 \le W \le 2 \cdot 10^6$, $0 \le p_i \le 2 \cdot 10^6$, $0 \le w_i \le 2 \cdot 10^6$ для всех $1 \le i \le n$. Все числа - целые.
- **Формат вывода / выходного файла (output.txt).** Выведите максимальное значение стоимости долей предметов, которые помещаются в сумку. Абсолютная погрешность между ответом вашей программы и оптимальным значением должно быть не более 10^{-3}. Для этого выведите свой ответ как минимум с четырьмя знаками после запятой (иначе ваш ответ, хотя и будет рассчитан правильно, может оказаться неверным из-за проблем с округлением).
- **Ограничение по времени.** 2 сек.

Решение:

Для решения данной задачи нужно отсортировать массив по удельной стоимости части товара. Для этого используем быструю сортировку по этому параметру. В исходном массиве хранятся пары элементов – стоимость товара и его вес.

```
import random

def randomized_quick_sort(A, f, l):
    if f < l:
        k = random.randint(f, l)
        A[f], A[k] = A[k], A[f]
        m1, m2 = part3(A, f, l)
        randomized_quick_sort(A, f, m1-1)
        randomized_quick_sort(A, m2+1, l)
```

```python
def part3(A, f, l):
    cur = A[f][0]/A[f][1]
    left = []
    mid = []
    right = []
    j = 0
    count = 0
    for i in range(f, l+1):
        if A[i][0]/A[i][1] > cur:
            left.append(A[i])
            j += 1
        elif A[i][0]/A[i][1] == cur:
            mid.append(A[i])
            count += 1
        else:
            right.append(A[i])
    A[f:l+1] = left + mid + right
    return f+j, f+j+count-1
```

Также нам понадобится функция для подсчета максимальной стоимости вмещающихся предметов. Пробегаясь по уже отсортированному массиву, в переменную ‘a’ минимум из оставшегося места в рюкзаке и вес текущего предмета. К ‘value’ добавляем стоимость вместившейся в рюкзак части предмета и уменьшаем вместимость рюкзака, пока она не станет равной нулю.

```python
def Knapsack(W, pw, n):
    value = 0
    for i in range(n):
        if W == 0:
            return value
        a = min(pw[i][1], W)
        value += pw[i][0]/pw[i][1]*a
        W -= a
    return value
```

В основном коде считываем входные данные, сортируем их по удельной стоимости и применяем функцию ‘Knapsack’, записывая ее результат в файл.

```python
pw = []
with open('input.txt') as f:
    n, W = map(int, f.readline().split())
    for _ in range(n):
        pw.append(list(map(int, f.readline().split())))

randomized_quick_sort(pw, 0, n-1)

with open('output.txt', 'w') as f:
    f.write(str(Knapsack(W, pw, n)))
```

Задание 3.

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.

Решение:

Для решения данной задачи нужно лишь отсортировать два данных массива и попарно перемножить i-ые элементы. В таком случае мы меньшие элементы перемножим с меньшими, а большие – с большими, получая максимальную сумму, которую и записываем в файл.

```python
with open('input.txt') as f:
    n = int(f.readline())
    a = list(map(int, f.readline().split()))
    b = list(map(int, f.readline().split()))
a.sort()
b.sort()

sum = 0
for i in range(n):
    sum += a[i] * b[i]

with open('output.txt', 'w') as f:
    f.write(str(sum))
```

Задание 8.

- **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.

- **Формат ввода / входного файла (input.txt).** В первой строке вводится натуральное число N - общее количество заявок на лекции. Затем вводится N строк с описаниями заявок - по два числа в каждом s_i и f_i для каждой лекции i. Гарантируется, что $s_i < f_i$. Время начала и окончания лекции - натуральные числа, не превышают 1440 (в минутах с начала суток).
- **Ограничения на входные данные.** $1 \le N \le 1000$, $1 \le s_i, f_i \le 1440$
- **Формат вывода / выходного файла (output.txt).** Выведите одно число – максимальное количество заявок на проведение лекций, которые можно выполнить.
- **Ограничение по времени.** 2 сек.
- **Ограничение по памяти.** 256 мб.

Решение:

В начале нам нужно отсортировать по возрастанию лекции по времени их начала, а те лекции, у которых оно одинаковое – по их концу. Для этого используем быструю сортировку сначала по первому индексу элемента, а в случае их равенства – по второму.

```
import random

def randomized_quick_sort(A, f, l):
    if f < l:
        k = random.randint(f, l)
        A[f], A[k] = A[k], A[f]
        m1, m2 = part3(A, f, l)
        randomized_quick_sort(A, f, m1-1)
        randomized_quick_sort_2(A, m1, m2)
        randomized_quick_sort(A, m2+1, l)
```

```python
def part3(A, f, l):
    cur = A[f][0]
    left = []
    mid = []
    right = []
    j = 0
    count = 0
    for i in range(f, l+1):
        if A[i][0] < cur:
            left.append(A[i])
            j += 1
        elif A[i][0] == cur:
            mid.append(A[i])
            count += 1
        else:
            right.append(A[i])
    A[f:l+1] = left + mid + right
    return f+j, f+j+count-1
```

```python
def randomized_quick_sort_2(A, f, l):
    if f < l:
        k = random.randint(f, l)
        A[f], A[k] = A[k], A[f]
        m1, m2 = part3_2(A, f, l)
        randomized_quick_sort_2(A, f, m1-1)
        randomized_quick_sort_2(A, m2+1, l)
```

```python
def part3_2(A, f, l):
    cur = A[f][1]
    left = []
    mid = []
    right = []
    j = 0
    count = 0
    for i in range(f, l+1):
        if A[i][1] < cur:
            left.append(A[i])
            j += 1
        elif A[i][1] == cur:
            mid.append(A[i])
            count += 1
        else:
            right.append(A[i])
    A[f:l+1] = left + mid + right
    return f+j, f+j+count-1
```

В основном коде считываем данные и сортируем их по вышеуказанному методу.

```python
lects = []
with open('input.txt') as f:
    N = int(f.readline())
    for _ in range(N):
        lects.append(list(map(int, f.readline().split())))
randomized_quick_sort(lects, 0, N-1)
```

Далее идем по отсортированному массиву. Если встречаем лекцию, которая заканчивается раньше, чем предыдущая, то кладем ее в ‘prev’. Если же встречаем лекцию, которая начинается позже конца предыдущей, увеличиваем счетчик и кладем эту лекцию в ‘prev’. В конце записываем счетчик в файл.

```
count = 1
prev = lects[0]
for i in range(1, N):
    if lects[i][0] > prev[0]:
        if lects[i][1] <= prev[1]:
            prev = lects[i]
        elif lects[i][0] >= prev[0]:
            count += 1
            prev = lects[i]

with open('output.txt', 'w') as f:
    f.write(str(count))
```

Задание 17.

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	.

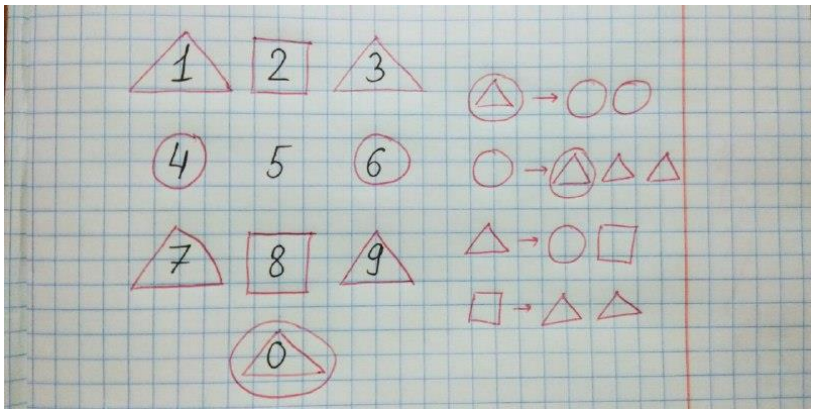
Напишите программу, определяющую количество телефонных номеров длины N, набираемых ходом коня. Поскольку таких номеров может быть очень много, выведите ответ по модулю 10^9.

- Формат ввода / входного файла (input.txt). Во входном файле записано одно целое число N.
- Ограничения на входные данные. $1 \le N \le 1000$.
- Формат вывода / выходного файла (output.txt). Выведите в выходной файл искомое количество телефонных номеров по модулю 10^9.
- Ограничение по времени. 1 сек.
- Ограничение по памяти. 256 мб.

Решение:

В начале считываем длину телефонного номера. Затем создаем два массива длиной 4. ‘prev’ сформирован согласно разделению цифр на категории (фото ниже).

```
with open('input.txt') as f:
    N = int(f.readline())
prev = [4, 2, 1, 0]
cur = [0 for _ in range(4)]
```



Далее, если длина номера равна 1, то ответом будет число 8, так как к исходным числам добавляется пятерка. В ином случае n-1 раз заполняем массив ‘cur’, согласно правилам, приведенным на фото выше, каждый раз в конце обновляя ‘prev’ на ‘cur’ и обнуляем ‘cur’. В конце выводим сумму элементов массива ‘prev’ – количество всевозможных номеров, по модулю 10^9.

```
if N == 1:
    res = 8
else:
    for i in range(N-1):
        cur[0] += prev[1] * 2 + prev[2] * 2
        cur[1] += prev[0] + prev[3] * 2
        cur[2] += prev[0]
        cur[3] += prev[1]
        prev = copy.copy(cur)
        cur = [0 for _ in range(4)]
    res = sum(prev) % 10 ** 9
with open('output.txt', 'w') as f:
    f.write(str(res))
```

Задание 18.

- Постановка задачи. Около университета недавно открылось новое кафе, в котором действует следующая система скидок: при каждой покупке более

чем на 100 рублей покупатель получает купон, дающий право на один бесплатный обед (при покупке на сумму 100 рублей и меньше такой купон покупатель не получает). Однажды вам на глаза попался прейскурант на ближайшие n дней. Внимательно его изучив, вы решили, что будете обедать в этом кафе все n дней, причем каждый день вы будете покупать в кафе ровно один обед. Однако стипендия у вас небольшая, и поэтому вы хотите по максимуму использовать предоставляемую систему скидок так, чтобы ваши суммарные затраты были минимальны. Требуется найти минимально возможную суммарную стоимость обедов и номера дней, в которые вам следует воспользоваться купонами.

- Формат ввода / входного файла (input.txt). В первой строке входного файла дается целое число n - количество дней. В каждой из последующих n строк дано одно неотрицательное целое число s_i – стоимость обеда в рублях на соответствующий день i.
- Ограничения на входные данные. $0 \le n \le 100$, $0 \le s_i \le 300$ для всех $0 \le i \le n$.
- Формат вывода / выходного файла (output.txt). В первой строке выдайте минимальную возможную суммарную стоимость обедов. Во второй строке выдайте два числа k_1 и k_2 – количество купонов, которые останутся у вас неиспользованными после этих n дней и количество использованных вами купонов соответственно. В последующих k_2 строках выдайте в возрастающем порядке номера дней, когда вам следует воспользоваться купонами. Если существует несколько решений с минимальной суммарной стоимостью, то выдайте то из них, в котором значение k_1 максимально (на случай, если вы когда-нибудь ещё решите заглянуть в это кафе). Если таких решений несколько, выведите любое из них.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 64 мб.

Решение:

Для решения данной задачи понадобится функция ‘dinam’, которая вычисляет минимальную сумму затрат в i-ый день и j оставшихся купонах. Так как количество купонов не может быть больше номера дня, но в этом случае сумма равна бесконечности. Также, если номер дня и количество купонов меньше 1, то сумма затрат 0, так они еще не произошли. В случае, если стоимость обеда меньше 101 и купонов меньше 1, сумма равна минимуму предыдущего дня на один купон больше или сумме предыдущего дня с тем же количеством купонов и стоимости обеда текущего дня. Рассмотрим случай, когда количество купонов больше 0. Параллельно, в некоторых случаях заполняется массив ‘A’, изначально составленный из ‘-1’. Если текущая ячейка имеет другое значение, то функция возвращает его. В ином случае, если

стоимость обеда больше 100, то результат равен минимуму предыдущего дня с одним купоном больше и суммы предыдущего дня с одним купоном меньше и стоимости текущего обеда. Иначе результат равен минимуму предыдущего дня с одним купоном больше и суммы предыдущего дня с тем же количеством купонов и стоимости текущего обеда.

```
def dinam(i, j):
    global A, prices
    if i >= 1:
        cost = prices[i-1]
    if j > i:
        return float('inf')
    else:
        if j <= 0:
            if i <= 0:
                return 0
            else:
                if cost <= 100:
                    res = min(dinam(i-1,j+1), dinam(i-1,j) + cost)
                else:
                    return dinam(i-1,j+1)
        else:
            if A[i][j] != -1:
                return A[i][j]
            if cost > 100:
                res = min(dinam(i-1,j+1), dinam(i-1,j-1) + cost)
            else:
                res = min(dinam(i-1,j+1), dinam(i-1,j) + cost)
        A[i][j] = res
        return res
```

Для реконструкции дней использования купонов используется стек. Номер дня добавляется в стек, если в этот день был использован купон. Этот факт определяется на основе индексов и сравнения значений функции ‘dinam’ в текущий и предыдущий день, и количествами купонов, зависящих от стоимости обеда в текущий день.

```
def reconstr(path, i, j):
    if j < i:
        global prices
        if i >= 1:
            cost = prices[i-1]
        if j <= 0:
            if i >= 1:
                if cost > 100:
                    path.append(i)
                    reconstr(path, i-1, j+1)
                else:
                    if dinam(i,j) == dinam(i-1,j+1):
                        path.append(i)
                        reconstr(path, i-1, j+1)
                    else:
                        reconstr(path, i-1, j)
        else:
            if cost <= 100:
                if dinam(i, j) == dinam(i-1, j+1):
                    path.append(i)
                    reconstr(path, i-1, j+1)
                else:
                    reconstr(path, i-1, j)
            else:
                if dinam(i, j) == dinam(i-1, j+1):
                    path.append(i)
                    reconstr(path, i-1, j+1)
                else:
                    reconstr(path, i-1, j-1)
```

В основном коде считываем данные и заводим вышеупомянутый массив ‘A’.

```
with open('input.txt') as f:
    n = int(f.readline())
    prices = []
    for i in range(n):
        prices.append(int(f.readline()))
A = [[-1 for _ in range(n+2)] for i in range(n+1)]
```

Далее для всех i, не больших n, находим минимальную сумму затрат в на момент последнего дня с i купонами и находим минимальное, при максимальном k_1. Далее, реконструируем дни траты купонов с помощью вышеописанной функции ‘reconstr’.

```
ans = float('inf')
for i in range(n+1):
    if dinam(n,i) <= ans:
        ans = dinam(n,i)
        k1 = i
path = deque()
reconstr(path, n, k1)
```

В конце записываем ответ в файл.

```
k2 = len(path)
with open('output.txt', 'w') as f:
    f.write(str(ans) + '\n')
    f.write(str(k1) + ' ' + str(k2) + '\n')

    while len(path) > 0:
        f.write(str(path.pop()) + '\n')
```

Задание 19.

- Постановка задачи. В произведении последовательности матриц полностью расставлены скобки, если выполняется один из следующих пунктов:
    - Произведение состоит из одной матрицы.
    - Оно является заключенным 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.
- Формат ввода / входного файла (input.txt). В первой строке входных данных содержится целое число n - количество матриц. В n следующих строк содержится по два целых числа a_i и b_i - количество строк и столбцов в i-той матрице соответственно. Гарантируется, что $b_i = a_i+1$ для любого $1 \le i \le n - 1$.
- Ограничения на входные данные. $1 \le n \le 400$, $1 \le a_i, b_i \le 100$ для всех $1 \le i \le n$.
- Формат вывода / выходного файла (output.txt). Выведите оптимальную расстановку скобок. Если таких расстановок несколько, выведите любую.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 64 мб.

Решение:

Считываем данные из файла так, чтобы каждый индекс встречался единожды.

```
m = []
with open('input.txt') as f:
    n = int(f.readline())
    if n % 2 == 1:
        for i in range(n):
            s = list(map(int, f.readline().split()))
            if i % 2 == 0:
                m.append(s[0])
                m.append(s[1])
    else:
        for i in range(n-1):
            s = list(map(int, f.readline().split()))
            if i % 2 == 0:
                m.append(s[0])
                m.append(s[1])
        s = list(map(int, f.readline().split()))
        m.append(s[1])
```

Создаем матрицы размера n, заполненные нулями. ‘C’ - матрица минимальных количеств операций. ‘S’ – матрица со ссылками на предыдущий элемент, из которого была взята минимальное количество операций для текущей последовательности матриц.

```
C = [[0 for i in range(n)] for _ in range(n)]
S = [[0 for i in range(n)] for _ in range(n)]
for i in range(n):
    S[i][i] = i
```

Далее заполняем массивы ‘C’ и ‘S’, получая значения из минимумов для предыдущей последовательностей матриц.

```
for s in range(1, n):
    for i in range(n-s):
        j = i+s
        minim = C[i][i] + C[i+1][j] + m[i] * m[i+1] * m[j+1]
        t = i
        for k in range(i+1, j):
            cur = C[i][k] + C[k+1][j] + m[i] * m[k+1] * m[j+1]
            if cur < minim:
                minim = cur
                t = k
        C[i][j] = minim
        S[i][j] = t
```

Затем, с помощью функции ‘printOpt’, восстанавливающей скобочное разделение матриц выводим ответ в файл. Данная функция работает с матрицей ссылок ‘S’, упомянутой выше.

```
res = ''
printOpt(S,0,n-1)
with open('output.txt', 'w') as f:
    f.write(res)
```

```
def printOpt(S,i,j):
    global res
    if i == j:
        res += 'A'
    else:
        res += '('
        printOpt(S,i,S[i][j])
        printOpt(S,S[i][j]+1,j)
        res += ')'
```

Задание 20.

- 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 «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.

- Формат входного файла (input.txt). В первой строке входного файла вводятся два натуральных числа: N – длина слова и K. Во второй строке записано слово S, состоящее из N строчных английских букв.
- Ограничения на входные данные. $1 \le N \le 5000$, $0 \le K \le N$.
- Формат выходного файла (output.txt). В выходной файл требуется вывести одно число – количество подслов слова S, являющихся почти палиндромами (для данного K).
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.

Решение:

Для выполнения данной задачи нам понадобится функция, будет проверять слово на то, что оно «почти палиндром».

```python
def almost_palindrom(s):
    global K
    changes = 0
    for i in range(len(s)//2):
        if s[i] != s[-(i+1)]:
            changes += 1
        if changes > K:
            return False
    return True
```

Считываем данные из файла.

```python
with open('input.txt') as f:
    N, K = map(int, f.readline().split())
    S = f.readline()
```

Далее проходимся по исходной строке. Каждый раз i-ый элемент становится центром палиндрома или элементов справа от его центра (если длина палиндрома четная). Наращивая строку справа и слева, проверяем, является ли она «почти палиндромом». Если нет, то обращаем соответствующий флаг в “False”. Если строка меньшей длины не является «почти палиндромом», то и строка с приращенными элементами справа и слева не является таковым. Таким образом, если оба флага ложны, то переходим в следующему элементу. Находя «почти палиндром», увеличиваем счетчик, которому изначально присвоено значение N, так как алгоритм не охватывает единичные строки, которые тоже являются «почти палиндромами». Записываем конечное значение счетчика в файл.

```python
res = N
for i in range(N):
    ind_left = i
    ind_right = i + 2
    flag1 = True
    flag2 = True
    while ind_left >= 0 and ind_right <= N:
        if flag1 and almost_palindrom(S[ind_left - 1:ind_right]) and ind_left - 1 >= 0:
            res += 1
        else:
            flag1 = False
        if flag2 and almost_palindrom(S[ind_left:ind_right]):
            res += 1
        else:
            flag2 = False
        if flag1 == False and flag2 == False:
            break
        ind_left -= 1
        ind_right += 1

with open('output.txt', 'w') as f:
    f.write(str(res))
```

Задание 21.

- **Постановка 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.

- **Формат входного файла (input.txt).** В первой строке входного файла находятся два натуральных числа N и M, а также символ R, означающий козырную масть. Во второй строке перечислены N карт, находящихся на руках у игрока. В третьей строке перечислены M карт, которые необходимо отбить. Все карты отделены друг от друга одним пробелом.
- **Ограничения на входные данные.** $N \leq 35$, $M \leq 4$, $M \leq N$.
- **Формат выходного файла (output.txt).** В выходной файл выведите «YES» в случае, если отбиться можно, либо «NO», если нельзя.
- **Ограничение по времени.** 1 сек.
- **Ограничение по памяти.** 16 мб.

Решение:

С помощью словаря ‘actions’ определим какими картами может быть побита конкретная по достоинству карта. Trump – случай, если карту другой масти нужно побить козырем.

```
actions = {'6':['7', '8', '9', 'T', 'J', 'Q', 'K', 'A'],
           '7':['8', '9', 'T', 'J', 'Q', 'K', 'A'],
           '8':['9', 'T', 'J', 'Q', 'K', 'A'],
           '9':['T', 'J', 'Q', 'K', 'A'],
           'T':['J', 'Q', 'K', 'A'],
           'J':['Q', 'K', 'A'],
           'Q':['K', 'A'],
           'K':['A'],
           'A':[None],
           'trump':['6', '7', '8', '9', 'T', 'J', 'Q', 'K', 'A']}
```

Считываем данные и распределяем карты по массивам их мастей.

```
with open('input.txt') as f:
    N, M, R = map(str, f.readline().split())
    my = list(map(str, f.readline().split()))
    enemy = list(map(str, f.readline().split()))
N, M = int(N), int(M)

S = []
C = []
D = []
H = []
for i in range(N):
    if my[i][1] == 'S':
        S.append(my[i][0])
    elif my[i][1] == 'C':
        C.append(my[i][0])
    elif my[i][1] == 'D':
        D.append(my[i][0])
    else:
        H.append(my[i][0])
```

Сначала пытаемся побить карту соперника наименьшей возможной картой той же масти.

```
for i in range(M):
    beaten = False
    suit = enemy[i][1]
    card = enemy[i][0]
    if suit == 'S':
        for j in actions[card]:
            if beaten == False:
                if j in S:
                    S.remove(j)
                    beaten = True
    elif suit == 'C':
        for j in actions[card]:
            if beaten == False:
                if j in C:
                    C.remove(j)
                    beaten = True
    elif suit == 'D':
        for j in actions[card]:
            if beaten == False:
                if j in D:
                    D.remove(j)
                    beaten = True
    else:
        for j in actions[card]:
            if beaten == False:
                if j in H:
                    H.remove(j)
                    beaten = True
```

Если же побить карту той же мастью не получилось, и эта масть козырная, то выводим ответ «NO».

```
    if beaten == False:
        if suit == R:
            with open('output.txt', 'w') as f:
                f.write('NO')
            exit()
```

Если же масть непобитой карты не козырная, то пытаемся побить ее наименьшей картой козырной масти.

```
        else:
            if R == 'S':
                for j in actions['trump']:
                    if beaten == False:
                        if j in S:
                            S.remove(j)
                            beaten = True
            elif R == 'C':
                for j in actions['trump']:
                    if beaten == False:
                        if j in C:
                            C.remove(j)
                            beaten = True
            elif R == 'D':
                for j in actions['trump']:
                    if beaten == False:
                        if j in D:
                            D.remove(j)
                            beaten = True
            else:
                for j in actions['trump']:
                    if beaten == False:
                        if j in H:
                            H.remove(j)
                            beaten = True
```

Если же карту так и не удалось побить, то записываем в файл «NO». В противном случае – «YES».

```
            if beaten == False:
                with open('output.txt', 'w') as f:
                    f.write('NO')
                exit()
with open('output.txt', 'w') as f:
    f.write('YES')
```

Вывод

В ходе проделанной лабораторной работы были изучены и применены на практике принципы и методы жадных алгоритмов и динамического программирования.