Санкт-Петербургский Национальный Исследовательский Университет
Информационных Технологий, Механики и Оптики

Факультет инфокоммуникационных технологий

Домашнее задание №5

Вариант №1

Выполнил:
Бацанова Е. А.
Проверил
Мусаев А.А.

Санкт-Петербург,
2024

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Целью данной работы является обобщение знаний об алгоритмах и их применение для решения различных задач. Для реализации решений для задач необходимо привлекать динамическое программирование, методы оптимизированного поиска, а также знания о формировании сложности того или иного алгоритма. Основными задачами, решаемыми в данной лабораторной работе были:

1) проверка результатов игры в крестики-нолики;
2) поиск элемента в построчно отсортированной по возрастанию матрице;
3) поиск расстановок ферзей на шахматной доске, таким образом, чтобы они не смогли атаковать друг друга;
4) поиск количества вариантов перемещения по лестнице, имеющей фиксированную длину, если совершать за одно перемещение не более чем фиксированное число шагов;
5) реализация трех стеков с использованием одномерного массива;
6) реализация экспоненциальный фильтра максимально коротким кодом;
7) поиск наименьшего пропущенного целого числа в несортированном массиве.

Задание 1

Задание: Разработайте алгоритм, проверяющий результат игры в крестики-нолики (3×3).

Решение:

Для реализации алгоритма создадим функцию *check_game_result(board)* (рис. 1). Данная функция проверяет выиграл ли кто-то из игроков с помощью вспомогательной функции *check_winner(board, player)*. Если победитель определен – он выводится на экран вместе с расположением крестиков и ноликов на доске, если нет – то игра либо закончилась ничьей, либо еще продолжается (о чем говорят незаполненные ячейки в массиве *board*).

Внутри функции *check_winner(board, player)* (рис. 2) происходит проверка на то, что строка, столбец или одна из диагоналей заполнены знаками типа *player*.

```
def check_game_result(board):
    if check_winner(board, 'X'):
        return 'X победил!'
    if check_winner(board, '0'):
        return '0 победил!'
    return 'Ничья!'
```

Рисунок 1 – Реализация функции, проверяющей результат игры в крестики-нолики (3×3)

```
def check_winner(board, player):
    for i in range(3):
        if all([cell == player for cell in board[i]]):
            return True
    for i in range(3):
        if all([board[j][i] == player for j in range(3)]):
            return True
    if all([board[i][i] == player for i in range(3)]) or all([board[i][2 - i] == player for i in range(3)]):
        return True
    return False
```

Рисунок 2 – Реализация функции, определяющей является ли игрок победителем в крестики-нолики (3×3)

```
['X', '0', '0']
['X', 'X', 'X']
['0', 'X', '0']
X победил!
```

Рисунок 3 – Вывод программы, проверяющей результат игры в крестики-нолики (3×3)

Задание 2

Задание: Для заданной матрицы M×N, в которой каждая строка и столбец отсортированы по возрастанию, напишите метод поиска элемента.

Решение:

Решение задачи реализуем через функцию *binary_search_in_matrix (matrix, target)* с помощью бинарного поиска по элементам матрицы. Для удобства транспонируем матрицу. Будем представлять эту транспонированную матрицу как единый массив длины M*N, а обращаться к его элементам как mas[m // M][m % M], где *m* – переменная, используемая в стандартном бинарном поиске по отсортированному массиву, каждый раз указывающая на середину рассматриваемого интервала значений. Если цикл завершается без нахождения значения *target*, то функция возвращает (-1, -1), в противном случае – координаты элемента в исходном массиве.

```
def binary_search_in_matrix(matrix, target):
    M, N = len(matrix), len(matrix[0])
    left, right = 0, M * N
    mas = [[matrix[j][i] for j in range(len(matrix))] for i in range(len(matrix[0]))]
    while left < right:
        m = (left + right) // 2
        if target > mas[m // M][m % M]:
            left = m + 1
        elif mas[m // M][m % M] == target:
            return m % M, m // M
        else:
            right = m
    return -1, -1
```

Рисунок 4 – Программа, ищущая элемент в матрицы с отсортированными по возрастанию строками

```
Матрица:
[1, 4, 7, 11]
[2, 5, 8, 12]
[3, 6, 9, 13]
Ищем число:  8
Элемент найден на позиции (1, 2)
```

Рисунок 5 – Вывод программы, ищущей элемент в матрицы с отсортированными по возрастанию строками

Задание 3

Задание: Напишите алгоритм, находящий все варианты расстановки восьми ферзей на шахматной доске размером 8×8 так, чтобы никакие две фигуры не располагались на одной горизонтали, вертикали или диагонали (учитываются не только две главные, но и все остальные диагонали)

Решение:

Реализуем алгоритм для решения задачи о расстановке восьми ферзей на шахматной доске размером 8×8 с помощью функции *queens(n, k)*. Функция принимает на вход размер доски *n*, число ферзей *k* и возвращает список возможных решений. Каждое решение представлено в виде списка строк, где каждая строка содержит координаты расположения ферзя на доске в стандартном виде.

Алгоритм основан на использовании рекурсивной функции *arrangement(row)*, которая пытается установить ферзя на каждой строке доски по очереди, проверяя при этом, что ферзь не атакует других ферзей (не находится на одной вертикали, горизонтали или диагонали с другими ферзями). Для проверки возможности поставить ферзя в заданную клетку доски используется функция *is_safe(row, col)*. После того как все ферзи были расставлены их текущее расположение добавляется в список решений *solutions*.

NB: программа способна искать возможные расположения ферзей при любом их количестве и любом размере доски – нет только 8×8

```python
def queens(n, k):
    board = [[0] * n for _ in range(n)]
    solutions = []
    alph = [chr(i) for i in range(ord('A'), ord('A') + n)]

    def is_safe(row, col):
        for i in range(row):
            if board[i][col] == 1:
                return False
        for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
            if board[i][j] == 1:
                return False
        for i, j in zip(range(row, -1, -1), range(col, n)):
            if board[i][j] == 1:
                return False
        return True

    def arrangement(row):
        if row == k:
            solution = []
            for r in range(n):
                for c in range(n):
                    if board[r][c] == 1:
                        solution.append(f'({alph[r]}, {c + 1})')
            solutions.append(solution)
        else:
            for col in range(n):
                if is_safe(row, col):
                    board[row][col] = 1
                    arrangement(row + 1)
                    board[row][col] = 0
    arrangement(0)
    return solutions
```

Рисунок 6 – Программа, реализующая поиск всех возможных расположений 8 ферзей на доске размером 8×8

```
Всего существует 92 позиции
['(A, 1)', '(B, 5)', '(C, 8)', '(D, 6)', '(E, 3)', '(F, 7)', '(G, 2)', '(H, 4)']
['(A, 1)', '(B, 6)', '(C, 8)', '(D, 3)', '(E, 7)', '(F, 4)', '(G, 2)', '(H, 5)']
['(A, 1)', '(B, 7)', '(C, 4)', '(D, 6)', '(E, 8)', '(F, 2)', '(G, 5)', '(H, 3)']
['(A, 1)', '(B, 7)', '(C, 5)', '(D, 8)', '(E, 2)', '(F, 4)', '(G, 6)', '(H, 3)']
['(A, 2)', '(B, 4)', '(C, 6)', '(D, 8)', '(E, 3)', '(F, 1)', '(G, 7)', '(H, 5)']
['(A, 2)', '(B, 5)', '(C, 7)', '(D, 1)', '(E, 3)', '(F, 8)', '(G, 6)', '(H, 4)']
['(A, 2)', '(B, 5)', '(C, 7)', '(D, 4)', '(E, 1)', '(F, 8)', '(G, 6)', '(H, 3)']
['(A, 2)', '(B, 6)', '(C, 1)', '(D, 7)', '(E, 4)', '(F, 8)', '(G, 3)', '(H, 5)']
['(A, 2)', '(B, 6)', '(C, 8)', '(D, 3)', '(E, 1)', '(F, 4)', '(G, 7)', '(H, 5)']
['(A, 2)', '(B, 7)', '(C, 3)', '(D, 6)', '(E, 8)', '(F, 5)', '(G, 1)', '(H, 4)']
['(A, 2)', '(B, 7)', '(C, 5)', '(D, 8)', '(E, 1)', '(F, 4)', '(G, 6)', '(H, 3)']
```

Рисунок 7 – Вывод программы, реализующей поиск всех возможных расположений 8 ферзей на доске размером 8×8

Задание 4

Задание: Ребенок поднимается по лестнице из n ступенек. За один шаг он может переместиться на одну, две или три ступеньки. Реализуйте метод, рассчитывающий количество возможных вариантов перемещения ребенка по лестнице.

Решение:

Решим задачу, реализовав функцию *count_ways(n)*, работающую на основе рекурсии. Функция принимает один аргумент *n* – количество ступеней на лестнице, будет меняться в каждой итерации, так как будем учитывать пройденные ступени. Если *n* в конкретную итерацию рекурсии меньше или равно 0, функция возвращает 0 – количество способов попасть на последнюю ступень при нулевой длине лестницы. Если *n* равно 1, функция возвращает 1 – число путей попасть из первой ступени на последнюю при длине лестницы 1 (способы: {1 шаг}). Аналогично – если *n* равно 2, функция возвращает 2 (способы: {1 шаг, 1 шаг} или {2 шага}) и если *n* равно 3, функция возвращает 4 (способы: {1 шаг, 1 шаг, 1 шаг}, {2 шага, 1 шаг}, {1 шаг, 2 шага} или {3 шага}). Во всех остальных случаях функция рекурсивно вызывает себя три раза для *n - 1*, *n - 2* и *n - 3* и возвращает сумму результатов этих вызовов.

```python
def count_ways(n):
    if n <= 0:
        return 0
    elif n == 1:
        return 1
    elif n == 2:
        return 2
    elif n == 3:
        return 4
    else:
        return count_ways(n - 1) + count_ways(n - 2) + count_ways(n - 3)
```

Рисунок 8 – Программа, рассчитывающая количество возможных вариантов перемещения по лестнице

```
Введите количество ступенек: 10
Количество способов перемещения по лестнице из 10 ступенек: 274
```

Рисунок 9 – Вывод программы, рассчитывающей количество возможных вариантов перемещения по лестнице

Задание 5

Задание: Опишите, как бы вы использовали один одномерный массив для реализации трех стеков.

Решение:

Попробуем реализовать три стека с помощью связного списка – базовой динамической структуры данных, состоящей из узлов, содержащих данные и ссылки на следующий и/или предыдущий узел списка. В данном случае в каждом узле хранится два элемента данных: data – значение данных, которое мы хотим хранить в текущем стеке и next – ссылка на следующую свободную позицию в массиве. Всего указателя будет 4 – один, указывающий на следующий свободный элемент, остальные – на три последних элемента каждого стека. Когда в стек добавляется элемент, он вставляется на место указателя на свободное место, сам указатель перемещается на следующую свободную позицию. Указатель на последний элемент стека встает на бывшее место свободного указателя. Когда мы удаляем элемент стека место добавляется в список свободных элементов. Указатель последнего элемента стека перенаправляется на прошлый элемент в стеке.

Это позволяет нам задействовать весь массив для добавления новых элементов в стек. Добавление элементов может происходить пока суммарный размер добавленных элементов не превысит длину массива.

```python
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None


class ThreeStacks:
    def __init__(self, stack_size):
        self.stack_size = stack_size
        self.stack_pointers = [None, None, None]
        self.free_pointer = None
        self.free_list = None

        for i in range(stack_size * 3):
            node = Node(None)
            node.next = self.free_list
            self.free_list = node

        self.free_pointer = self.free_list

    def push(self, stack_num, data):
        if self.is_full():
            print("Stack is full")
            return None

        new_node = self.free_pointer
        self.free_pointer = self.free_pointer.next
        new_node.data = data

        new_node.next = self.stack_pointers[stack_num]
        self.stack_pointers[stack_num] = new_node

    def pop(self, stack_num):
        if self.is_empty(stack_num):
            print("Stack is empty")
            return None

        popped_node = self.stack_pointers[stack_num]
        self.stack_pointers[stack_num] = popped_node.next

        popped_data = popped_node.data
        popped_node.data = None
        popped_node.next = self.free_pointer
        self.free_pointer = popped_node

        return popped_data

    def is_full(self):
        return self.free_pointer is None

    def is_empty(self, stack_num):
        return self.stack_pointers[stack_num] is None
```

Рисунок 9 – Реализация 3 стеков из одномерного массива

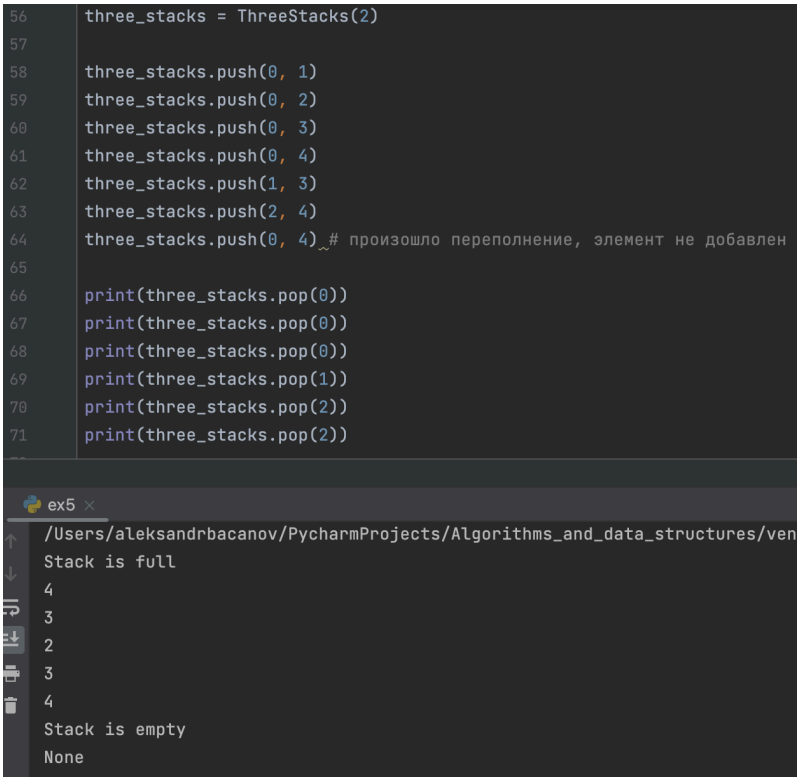

Рисунок 9 – Пример использования 3 стеков из одномерного массива – удаление, добавление, переполнение

Задание 6

Задание: Напишите максимально короткий код для экспоненциального фильтра.

Решение:

Реализуем экспоненциальное сглаживание по стандартному алгоритму:

➔ $s_t = c_t$ при $t = 1$;

➔ $s_t = s_{t-1} + \alpha(c_t - s_{t-1})$ в остальных случаях.

```
def exponential_filter(data, alpha):
    result = [data[0]]
    for i in range(1, len(data)):
        result.append(result[i - 1] + alpha * (data[i] - result[i - 1]))
    return result
```

Рисунок 10 – Программа. реализующая экспоненциальное сглаживание (максимально короткий код)

```
Изначальный ряд: 10 20 30 40 50
Ряд после экспоненциального сглаживания: 10 12.0 15.6 20.48 26.384
```

Рисунок 11 – Пример вывода программы, реализующей экспоненциальное сглаживание

Задание 7

Задание: Дан неотсортированный массив целых чисел. Верните наименьшее пропущенное целое число. Алгоритм должен выполняться за время O(n).

Решение:

Поиск наименьшего пропущенного целого числа реализуется функцией *smallest_missing_integer(arr)*. Данная программа принимает на вход список целых чисел *arr*, затем программа находит максимальное и минимальное значение в этом списке. Затем создаются два списка длиной – список *minus* (состоит из одного 0 если в *arr* нет отрицательных элементов и имеющий длину на 1 большую, чем модуль минимального числа, в обратном случае) и *plus* (пустой если в *arr* нет положительных элементов и имеющий длину на 1 большую максимального числа массива в обратном случае). Все эти операции имеют сложность O(1). Далее программа проходит по всем элементам списка *arr*: если число меньше или равно 0, то оно записывается в minus с индексом -num, если число больше 0, то в plus с индексом *num*. Эта операция имеет сложность O(n), где n – число элементов в массиве.

После этого программа проверяет наличие 0 в minus или plus[1:]. Если оно есть, то возвращаем позицию первого нуля в списке plus[1:], прибавляя 1, так как в plus хранятся значения начиная с 0, или позицию в инвертированном списке minus. Если 0 нет в обоих этих списках – выводим первое число после максимального числа массива. Данные операции также имеют сложность O(1).

В итоге общая сложность равняется O(n).

```python
def smallest_missing_integer(arr):
    max_num, min_num = max(arr), min(arr)
    minus = [0] * (- min_num + 1)
    plus = [0] * (max_num + 1)
    for num in arr:
        if num <= 0:
            minus[-num] = 1
        else:
            plus[num] = 1
    if 0 in minus:
        return minus[::-1].index(0) + min_num
    if 0 in plus:
        if plus.index(0) != 0:
            return plus.index(0)
    return max_num + 1
```

Рисунок 12 –Программа, ищущая минимальное пропущенное целое число (сложность O(n))

```
Наименьшее пропущенное целое число в массиве [5, 4, 2, -1, -3, 0]: -2
```

Рисунок 13 – Вывод программы, ищущей минимальное пропущенное целое число (сложность O(n))

ЗАКЛЮЧЕНИЕ

В результате выполнения работы были разработаны решения различных задач с использованием изученных ранее алгоритмов.

Бинарный поиск был использован в задаче о поиске элемента в построчно отсортированной по возрастанию матрице.

Динамическое программирование было использовано в задачах о поиске расстановок ферзей на шахматной доске, таким образом, чтобы они не смогли атаковать друг друга, поиске количества вариантов перемещения по лестнице, имеющей фиксированную длину, если совершать за одно перемещение не более чем фиксированное число шагов.

Рекуррентные соотношения были использованы при реализации задачи реализации экспоненциального фильтра максимально коротким кодом.

Понятие связного списка, стека и узлов было использовано в задаче о реализации трех стеков с использованием одномерного массива.

Разработанные алгоритмы были успешно протестированы и показали свою эффективность и работоспособность. Таким образом, выполнение поставленных задач позволило не только углубить знания и практические навыки в области алгоритмов, но и применить их при решении практических и теоретических задач.

СПИСОК ЛИТЕРАТУРЫ

1) Wikipedia. Связный список. [Электронный ресурс] – (Дата последнего обращения 19.05.2024);
2) Wikipedia. Экспоненциальное сглаживание. [Электронный ресурс] – (Дата последнего обращения 19.05.2024).

ПРИЛОЖЕНИЕ

Для удобства все файлы выгружены на GitHub: https://github.com/kathykkKk/Algorithms-and-Data-Structures-ICT.git