**Формальные языки и алгоритмы**

**преподаватель: Масленников Д. А.**

**выполнено студентом 23-САИ-1 Ламовым Вячеславом**

# **Глава 1.**

## Глава 1. Задачи на 4.

* 1. Имеется отсортированный список из 128 имен, и вы ищете в нем значение методом бинарного поиска. Какое максимальное количество проверок для этого может потребоваться?

**Ответ:** Воспользуемся бинарным поиском. Худшим для нас случаем будет, когда искомое значение будет на первой или последней позиции. Предположим, что это значение находиться на 128ой позиции. Тогда мы будем делить список на равные отрезки до тех пор, пока не дойдем до 128:

1…128(mid=64), 64…128(m=96), 96…128(m=112), 112…128(m=120), 120…128(m=124), 124…128(m=126), 126…128(mid=127), 128

Как итог 7 попыток.

* 1. Предположим, размер списка увеличился вдвое. Как изменится максимальное количество проверок?  
     **Ответ:** Аналогичной номеру 1.1 логикой получаем ответ: 8 попыток
  2. Известна фамилия, нужно найти номер в телефонной книге.  
     **Ответ:** O(log(n)), так как список фамилий является отсортированным, то можно воспользоваться бинарным поиском.
  3. Известен номер, нужно найти фамилию в телефонной книге.  
     **Ответ:** O(n) поскольку номера не отсортированы в телефонной книге, в худшем случае придется пройтись по каждой строчке
  4. Нужно прочитать номера всех людей в телефонной книге.  
     **Ответ:** O(n, потому придётся пройтись по каждому элементу
  5. Нужно прочитать телефоны всех людей, фамилии которых начинаются с буквы «А».  
     **Ответ:** O(n), потому что константные множители не влияют на сложность алгоритма

## Глава 1. Задачи на 5.

Имеется массив объектов, отсортированный по 3 и более критериям. Осуществить в нем бинарный поиск.

Стандартная реализация бинарного поиска на языке Python справляется в поставленной задачей и не требует дополнительных корректировок. Происходит это по той причине, что Python умеет сравнивать итерируемые объекты. На примере кортежей и списков это работает так: элементы сравниваются друг с другом по порядку, если один из элементов больше весь список или кортеж считается больше сравниваемого, если элементы равны, по сравниваются следующие по порядку. Если все сравниваемые элементы списка или кортежа оказались равными, но один из объектов длиннее (содержит в себе больше элементов), то ему присуждается звание большего.

Несколько тестов

Представленный код

print(f'1. {[2004, 11, 21] < [2005, 1, 18]}')  
print(f"2. {[2005, 1, 18] > [2005, 1, 19]}")  
print(f"3. {[2005, 1, 18, 6] > [2005, 1, 19]}")  
print(f"4. {[2005, 1, 19] < [2005, 1, 19, 1]}")  
print(f"5. {('winter', 'spring', 'summer') == ('winter', 'spring', 'summer')}")  
print(f"6. {('winter', 'spring', 'summer') > ('winter', 'spring', 'summer', 'fall')}")  
print(f"7. {('winter', 'spring', 'summer') < ('winter', 'spring', 'summer', 'fall')}")

после выполнения выведет

1. True

2. False

3. False

4. True

5. True

6. False

7. True

Важный момент, чтобы тип сравниваемых элементов был одинаковым, иначе это приведет к ошибке.

print(f'1. {[2004, 11, 21] < [2004, 11, "fall"]}')

выводит

File "%%%", line 29, in <module>

print(f'1. {[2004, 11, 21] < [2004, 11, "fall"]}')

TypeError: '<' not supported between instances of 'int' and 'str'

Демонстрация работы бинарного поиска на примере отсортированного списка кортежей дней рождений

def binary\_search(list, item):  
 low = 0  
 high = len(list) - 1  
 counter = 0  
 while low <= high:  
 mid = (low + high) // 2  
 guess = list[mid]  
 if guess == item:  
 return mid  
 if guess > item:  
 counter += 1  
 high = mid - 1  
 else:  
 counter += 1  
 low = mid + 1  
  
birthdays = [(2004, 8, 22), (2004, 11, 21), (2004, 12, 19), (2005, 1, 18), (2005, 3, 11), (2005, 8, 18), (2005, 8, 19)]

print(f'Индекс искомого элемента {binary\_search(birthdays, (2004, 12, 19))}')

Выводит

Индекс искомого элемента 2

# **Глава 2.**

## Глава 2. Задачи на 4.

* 1. Какую структуру использовать для приложения управления финансами — массив или список?  
     **Ответ:** Предпочтительнее использовать списки, так как операция вставки в списках имеет время выполнения O(1). А в условии задачи указано, что выполняется множество операций вставки и относительно немного операций чтения. Выбор очевиден.
  2. Приложение для приема заказов от посетителей ресторана. Приложение должно хранить список заказов. Официанты добавляют заказы в список, а повара читают заказы из списка и выполняют их. Заказы образуют очередь: официанты добавляют заказы в конец очереди, а повар берет первый заказ из очереди и начинает готовить. Какую структуру данных вы бы использовали?  
     **Ответ:** Я бы выбрал использовать связанные списки, так как время выполнения операции вставки O(1), а так как повара читают первый заказ из списка, то чтение будет тоже O(1).
  3. Как бы вы реализовали список пользователей Facebook — в виде массива или связанного списка?  
     **Ответ:** Я бы предпочел массив, так как на чтение в массиве затрачивается O(1), что будет большим преимуществом, когда нужно мгновенно считать средний элемент.
  4. Какими недостатками обладает массив для выполнения вставки?  
     **Ответ:** Минусом использования массива будет то, что в какой-то момент может закончиться выделенное под него место. А также, учитывая тот факт, что пользователи довольно часто создают новые учетные записи, время выполнения записи в массиве равное O(n) будет серьезным минусом. А факт, что новый элемент добавляется в конец массива, вынудит каждую запись сортировать массив заново.
  5. Сравните эту гибридную структуру данных с массивами и связанными списками. Будет она быстрее или медленнее каждой исходной структуры при поиске и вставке?  
     **Ответ:** Поиск окажется медленнее, чем в случае с массивами, но быстрее, чем для связанных списков. Вставка – быстрее, чем у массивов, и такая же, как у списков.

## Глава 2. Задачи на 5.

# **Глава 3.**

## Глава 3. Задачи на 4.

1. Что можно сказать о текущем состоянии программы на основании этого стека вызовов?  
   **Ответ:**

* сначала вызывается функция greet для переменной name = maggie;
* затем функция greet вызывает функцию greet2 для переменной name = maggie;
* на этой стадии функция greet находится в незавершенном, при­остановленном состоянии;
* текущим вызовом функции является вызов greet2;
* после завершения этого вызова функция greet продолжит выпол­нение.

1. А что произойдет со стеком при бесконечном выполнении рекурсии?  
   **Ответ:** Стек будет расти бесконечно, пока программа не выдаст ошибку о переполнении стека.

Последовательности Фибоначчи на основе рекурсии.

def fibonacci(n):  
 if n == 0 or n == 1:  
 return (0, 1)[n]  
 else:  
 return fibonacci(n-1) + fibonacci(n-2)

При n = 0 или 1 возвращает 0 или 1 соответственно, в остальных случаях уходит в рекурсию. Вычислительная сложность O().

Реализация сложения(вычетанания) столбиком.  
def stolbik(num1, num2):  
 if len(num1) != len(num2):  
 result = [0] + max(num1, num2, key=len)  
 small = min(num1, num2, key=len)  
 else:  
 result = [0] + num1  
 small = num2  
 small\_len = len(small)  
 for i in range(1, small\_len + 1):  
 temp = result[-i] + small[-i]  
 result[-i] = temp % 10  
 result[-i-1] = result[-i-1] + temp // 10  
 while True:  
 if result[0] != 0 or len(result) == 1:  
 break  
 else:  
 del result[0]  
 return result

Реализация последовательности Фибоначчи на основе сложения столбиком.

def fibonacciStolbik(n):  
 if n == [0] or n == [1]:  
 return ([0], [1])[n[0]]  
 else:  
 return stolbik(fibonacciStolbik(stolbik(n, [-1])), fibonacciStolbik(stolbik(n, [-2])))

## Глава 3. Задачи на 5.

Последовательности Фибоначчи на основе длинной арифметики

def fibanacci(n):  
 fst, snd = 0, 1  
 for i in range(n):  
 fst, snd = snd, fst + snd  
 return fst

Вычислительная сложность алгоритма – линейная O(n)

Реализация последовательности Фибоначчи на основе длинной арифметики и сложения столбиком.

def fibanacciStolbik(n):  
 fst, snd = [0], [1]  
 for i in range(n):  
 fst, snd = snd, stolbik(fst, snd)  
 return fst

24000

fibonacci: 0:00:00.010002

fibonacciStolbik: 0:00:24.733636

48000  
fibonacci: 0:00:00.03200  
fibonacciStolbik: 0:01:37.969486

10\*\*5  
fibonacci: 0:00:00.107023  
fibonacciStolbik: 0:07:10.439200

10\*\*6  
fibonacci: 0:00:09.109041  
fibonacciStolbik: >0:20:00.000000

# **Глава 4.**

## Глава 4. Задачи на 4.

1. Напишите код для функции sum  
   **Ответ:**def sum(list):  
    if list = []:  
    return 0  
    return list[0] + sum(list[1:])

если список пустой возвращает ноль, в противном случае складывает первые значения списка, уходя в рекурсию

1. Напишите рекурсивную функцию для подсчета элементов в списке.  
   **Ответ:**def count(list):  
    if list = []:  
    return 0  
    return 1 + 1\*count(list[1:])

Пока список не будет пустым рекурсивно возвращает 1

1. Найдите наибольшее число в списке  
   **Ответ:**def max(list):  
    if len(list) == 0:  
    return None  
    if len(list) == 1:  
    return list[0]  
    else:  
    max\_num = max(list[1:])  
    return lst[0] if lst[0] > max\_num else max\_num
2. Сможете ли вы определить базовый и рекурсивный случай для бинарного поиска? **Ответ:** Базовый, когда массив для поиска из одного элемента. В рекурсивном массив делится пополам, одна отбрасывается, для другой выполняется бинарный поиск.
3. Вывод значения каждого элемента массива. **Ответ:** O(n), придется пройти по всему массиву.
4. Удвоение значения каждого элемента массива. **Ответ:** O(n), придется пройти по всему массиву.
5. Удвоение значения только первого элемента массива. **Ответ:** O(1)
6. Создание таблицы умножения для всех элементов массива.  
   **Ответ:** O(n2)

## Глава 4. Задачи на 5.

*#сортировка подсчетом массива, в котором значения могут быть от 0 до 100*def counting\_search(some\_list):  
 counting\_list = [0] \* 100  
 for elem in some\_list:  
 counting\_list[elem] += 1  
 result = []  
 for num in range(100):  
 if counting\_list[num] != 0:  
 result.extend([num]\*counting\_list[num])  
 return result  
print(\*counting\_search([2, 4, 1, 5, 6, 5, 6]))

Выводит

1 2 4 5 5 6 6

Сложность алгоритма линейная O(n)

10\*\*4

counting\_sort: 0:00:00.002001

quicksort: 0:00:00.015003

10\*\*5

counting\_sort: 0:00:00.009002  
quicksort: 0:00:00.133032

10\*\*6  
counting\_sort: 0:00:00.072016  
quicksort: 0:00:01.805406

10\*\*7  
counting\_sort: 0:00:00.848191  
quicksort: 0:00:27.889497

10\*\*8  
counting\_sort: 0:00:09.088042  
quicksort: 0:06:20.199689

# **Глава 5.**

## Глава 5. Задачи на 4.

1. f(x) = 1  
   **Ответ:** является последовательной, при любых входных данных возвращает один и тот же результат - единицу
2. f(x) = rand() **Ответ:** не является последовательной, так как каждый раз возвращает случайное значение
3. f(x) = next\_empty\_slot() **Ответ:** Не является последовательной, так как индекс следующего пустого элемента обязательно больше предыдущего, следовательно функция выводит разный результат.
4. f(x) = len(x) **Ответ:** Последовательная функция, так как длина при одних и тех же значениях не изменяется.
5. Первая функция возвращает «1» для любого входного значения.
6. Вторая функция возвращает длину строки в качестве индекса.
7. Третья функция возвращает первый символ строки в качестве индекса. Таким образом, все строки, начинающиеся с «a», хешируются в одну позицию, все строки, начинающиеся с «b», — в другую и т. д.
8. Четвертая функция ставит в соответствие каждой букве простое число: a = 2, b = 3, c = 5, d = 7, e = 11 и т. д. Для строки хеш-функцией становится остаток от деления суммы всех значений на размер хеша. Например, если размер хеша равен 10, то для строки «bag» будет вычислен индекс 3 + 2 + 17 % 10 = 22 % 10 = 2.

В каком из этих примеров хеш-функции будут обеспечивать хорошее распределение? Считайте, что хеш-таблица содержит 10 элементов.

1. Телефонная книга, в которой ключами являются имена, а значениями — номера телефонов. Задан следующий список имен: Esther, Ben, Bob, Dan. **Ответ:** Четвертая функция будет обеспечивать хорошее распределение, потому что Ben: (3 + 11 + 43)%10 = 7, Bob: (3 + 47 + 3)%10 = 3, Dan: (7 + 2 + 43)%10 = 2, Esther: (11 + 67 + 71 + 19 + 11 + 61)%10 = 0, соответственно каждому имени свой хеш.
2. Связь размера батарейки с напряжением. Размеры батареек: A, AA, AAA, AAAA. **Ответ:** Вторая функция будет обеспечивать хорошее распределение, потому что размер батарейки отличается количеством А, следовательно для каждому размеру будет соответствовать свой хеш. А также четвертая функция будет обеспечивать хорошее распределение, так как для А будет хеш 2, для АА – 4, ААА – 6, АААА – 8.
3. Связь названий книг с именами авторов. Названия книг: «Maus», «Fun Home», «Watchmen». **Ответ:** Третья функция будет обеспечивать хорошее распределение, потому что название каждой книги, начинается с разной буквы. Четвертая функция будет обеспечивать хорошее распределение, так как для Maus: (41+2+73+67)%10 = 3, Fun Home: (13 + 73 + 43 + 19 + 47 + 41 + 11)%10 = 7, Watchmen: (83 + 2 + 71 + 5 + 19 + 41 + 11 + 43)%10 = 5.

## Глава 5. Задачи на 5.

Поиск

|  |  |
| --- | --- |
|  | список |
| 19\*10^7 | 0.0009937286376953125 |
| 20\*10^7 | 0.0011780261993408203 |
| 25\*10^7 | 0.0010006427764892578 |
| 28\*10^7 | 0.0010004043579101562 |
| 30\*10^7 | 0.0017600059509277344 |
| 33\*10^7 | 0.0010004043579101562 |
| 35\*10^7 | 0.0009996891021728516 |

|  |  |  |
| --- | --- | --- |
|  | массив | словарь |
| 10\*10^8 | 0.0003275871276855469 | 0.0011267662048339844 |
| 13\*10^8 | 0.002653360366821289 | 0.0018563270568847656 |
| 15\*10^8 | 0.003626585006713867 | 0.0010001659393310547 |
| 19\*10^8 | 0.004001140594482422 | 0.001001119613647461 |
| 20\*10^8 | 0.0040013790130615234 |  |

Вставка

|  |  |
| --- | --- |
|  | список |
| 10^5 | 0.0009999275207519531 |
| 10^6 | 0.002000570297241211 |
| 10^7 | 0.017003536224365234 |
| 10^8 | 0.1540358066558838 |
| 10^9 | 12.927921056747437 |

|  |  |
| --- | --- |
|  | массив |
| 10^5 | 0.0009999275207519531 |
| 10^6 | 0.002000570297241211 |
| 10^7 | 0.017003536224365234 |
| 10^8 | 0.1540358066558838 |
| 10^9 | 12.927921056747437 |

# **Глава 6.**

## Глава 6. Задачи на 4.

6.1 Найдите длину кратчайшего пути от начального до конечного узла.

from collections import deque  
  
  
graph = {"S": ("A", "C"), "A": ("B", "F"), "C": ("B", "D"), "D": ("F",), "B": (None,)}  
search\_queue = deque()  
search\_queue += graph["S"]  
destination = "F"  
path = 0  
temp = 0  
levels = [len(graph["S"]), 0]  
while search\_queue:  
 point = search\_queue.popleft()  
 temp += 1  
 if temp == levels[path]:  
 path += 1  
 temp = 0  
 levels.append(0)  
 if point == destination:  
 break  
 else:  
 search\_queue += graph[point]  
 levels[path+1] = levels[path+1] + len(graph[point])  
print(path)

Вывод:

2

6.2 Найдите длину кратчайшего пути от «cab» к «bat».

from collections import deque  
  
  
graph = {"CAB": ("CAT", "CAR"), "CAR": ("BAR", "CAT"), "CAT": ("MAT", "BAT"), "BAR": ("BAT",), "MAT": ("BAT",)}  
search\_queue = deque()  
search\_queue += graph["CAB"]  
destination = "BAT"  
path = 0  
temp = 0  
levels = [len(graph["CAB"]), 0]  
while search\_queue:  
 point = search\_queue.popleft()  
 temp += 1  
 if temp == levels[path]:  
 path += 1  
 temp = 0  
 levels.append(0)  
 if point == destination:  
 break  
 else:  
 search\_queue += graph[point]  
 levels[path+1] = levels[path+1] + len(graph[point])  
print(path)

Вывод:

2

Добавить дополнительную структуру, в которой устанавливается соответствие между именами людей и фактом того, что они являются продавцами манго.

class MangoSeller():

def \_\_init\_\_(self):  
 self.mango\_sellers\_list = ["semen bakin", "slava kpps", "daniil guskov"]

def person\_is\_seller(self, name):  
 return name in self.mango\_sellers\_list  
   
 def add\_seller(self, name):  
 if name not in self.mango\_sellers\_list:  
 self.mango\_sellers\_list.append(name)

6.3 Для каждого из следующих трех списков укажите, действителен он или недействителен.

Ответ: Ответы: A — недействителен; B — действителен; С — недействителен.

6.4 Постройте действительный список для этого графа.

Ответ: 1 — Проснуться; 2 — Сделать зарядку; 3 — Принять душ; 4 — Почистить зубы; 5 — Одеться; 6 — Упаковать обед; 7 — Позавтракать.

6.5 Какие из следующих графов также являются деревьями?

Ответ: А — дерево; B — не дерево; C — дерево.

## Глава 6. Задача на 5.

from time import sleep  
from collections import deque  
from random import \*  
  
  
'''Поле лабиринта задаётся нулями и единицами, где ноль выступает в роли стены, а единица - пути'''  
  
test2\_labyrinth = [  
 [0, 0, 1, 1, 1, 1, 0, 1],  
 [0, 1, 1, 0, 0, 0, 0, 0],  
 [1, 0, 1, 0, 0, 0, 1, 1],  
 [1, 1, 1, 1, 0, 1, 0, 1],  
 [0, 0, 1, 1, 1, 0, 1, 0],  
 [0, 0, 1, 0, 1, 1, 1, 0],  
 [1, 0, 1, 0, 0, 1, 0, 1],  
 [0, 0, 1, 0, 0, 1, 1, 0],  
]  
  
test1\_labyrinth = [  
 [1, 0, 0, 0],  
 [1, 1, 1, 0],  
 [0, 0, 1, 1],  
 [0, 1, 1, 0],  
]  
  
  
class Labyrinth:  
  
 def \_\_init\_\_(self, labyrinth):  
 self.width = 8  
 self.height = 8  
 self.labyrinth = [[randint(0, 1) for \_ in range(self.width)] for \_ in range(self.height)]  
 *#self.labyrinth = labyrinth* def show\_labyrinth(self):  
 coors = list(str(i) for i in range(self.width))  
 print(f'\* {" ".join(coors)}')  
 print('—'\*2\*(self.width+1))  
 for line, coor in zip(self.labyrinth, coors):  
 print(f'{coor}|', end='')  
 print(\*line)  
 sleep(0.3)  
  
 def check\_start\_n\_end(self, start, end):  
 if self.labyrinth[start[0]][start[1]] \* self.labyrinth[end[0]][end[1]] == 0:  
 return -2  
 return True

"""В функции create\_connections проходимся по всем вершинам графа и если он равна 0, то проверяем соседние на равенство единице.  
 Если соседняя равна единице, то добавляем координату в список соседов в словаре, где ключи это координаты вершины"""  
 def create\_connections(self):  
 c = ((-1, 0), (1, 0), (0, 1), (0, -1))  
 res = {}  
 for i in range(self.height):  
 for j in range(self.width):  
 if self.labyrinth[i][j] == 1:  
 for neighbor in c:  
 x = i + neighbor[0]  
 y = j + neighbor[1]  
 if 0 <= x <= self.width - 1 and 0 <= y <= self.width - 1 and self.labyrinth[x][y] == 1:  
 res[(i, j)] = res.get((i, j), []) + [(x, y)]  
 return res  
  
 def search(self, start, destination):  
 graph = self.create\_connections()  
 search\_queue = deque()  
 try:  
 search\_queue += graph[start]  
 except KeyError:  
 return -1  
 searched = [start]  
 path = 1  
 temp = 0  
 levels = [len(graph[start]), 0]  
 while search\_queue:  
 point = search\_queue.popleft()  
 temp += 1  
 levels[path] = levels[path] + len(set(graph[point]) - set(searched))  
 if point == destination:  
 return path  
 else:  
 search\_queue += list(set(graph[point]) - set(searched))  
 searched.append(point)  
 if temp == levels[path-1]:  
 path += 1  
 temp = 0  
 levels.append(0)  
 return -1

class UI:  
  
 @staticmethod  
 def generate():  
 for \_ in range(3):  
 print('----------------')  
 print('---GENERATING---')  
 sleep(0.4)  
  
 @staticmethod  
 def get\_coordinate():  
 start = input("Введите координаты начала YX\n")  
 end = input("Введите координаты конца YX\n")  
 return (int(start[0]), int(start[1])), (int(end[0]), int(end[1]))  
  
  
class Main:  
  
 def \_\_init\_\_(self):  
 self.lab = Labyrinth(test1\_labyrinth)  
 self.UI = UI()  
  
 def run(self):  
 self.UI.generate()  
 self.lab.show\_labyrinth()  
 while True:  
 start, end = self.UI.get\_coordinate()  
 if self.lab.check\_start\_n\_end(start, end) == -2:  
 print(-2)  
 else:  
 print(self.lab.search(start, end))  
  
  
Main().run()

# **Глава 7.**

## Глава 7. Задачи на 4.

7.1 Граф А

graph = {"start": {"a": 2, "b": 5}, "a": {"b": 8, "d": 7},"b": {"c": 4, "d": 2},  
 "c": {"end": 3, "d": 6}, "d": {"end": 1}, "end": {}}  
infinity = float("inf")  
costs = {"a": 2, "b": 5, "c": infinity, "d": infinity, "end": infinity}  
parents = {"a": "start", "b": "start", "c": "b", "d": None, "end": None}  
processed = []  
  
  
def find\_lowest\_cost\_node(costs):  
 lowest\_cost = float("inf")  
 lowest\_cost\_node = None  
 for node in costs:  
 cost = costs[node]  
 if cost < lowest\_cost and node not in processed:  
 lowest\_cost = cost  
 lowest\_cost\_node = node  
 return lowest\_cost\_node  
  
  
node = find\_lowest\_cost\_node(costs)  
while node is not None:  
 cost = costs[node]  
 neighbors = graph[node]  
 for n in neighbors.keys():  
 new\_cost = cost + neighbors[n]  
 if costs[n] > new\_cost:  
 costs[n] = new\_cost  
 parents[n] = node  
 processed.append(node)  
 node = find\_lowest\_cost\_node(costs)  
  
print(costs["end"])

Граф B

graph = {"start": {"a": 10}, "a": {"b": 20}, "b": {"c": 1, "end": 30},  
 "c": {"a": 1}, "end": {}}  
infinity = float("inf")  
costs = {"a": 10, "b": infinity, "c": infinity, "end": infinity}  
parents = {"a": None, "b": "a", "c": "b", "end": "b"}  
processed = []

Код аналогичный

Проблема заключается в том, что алгоритм Дейкстры работает только с направленным ациклическим графом, а тут встречается цикл.

Граф C

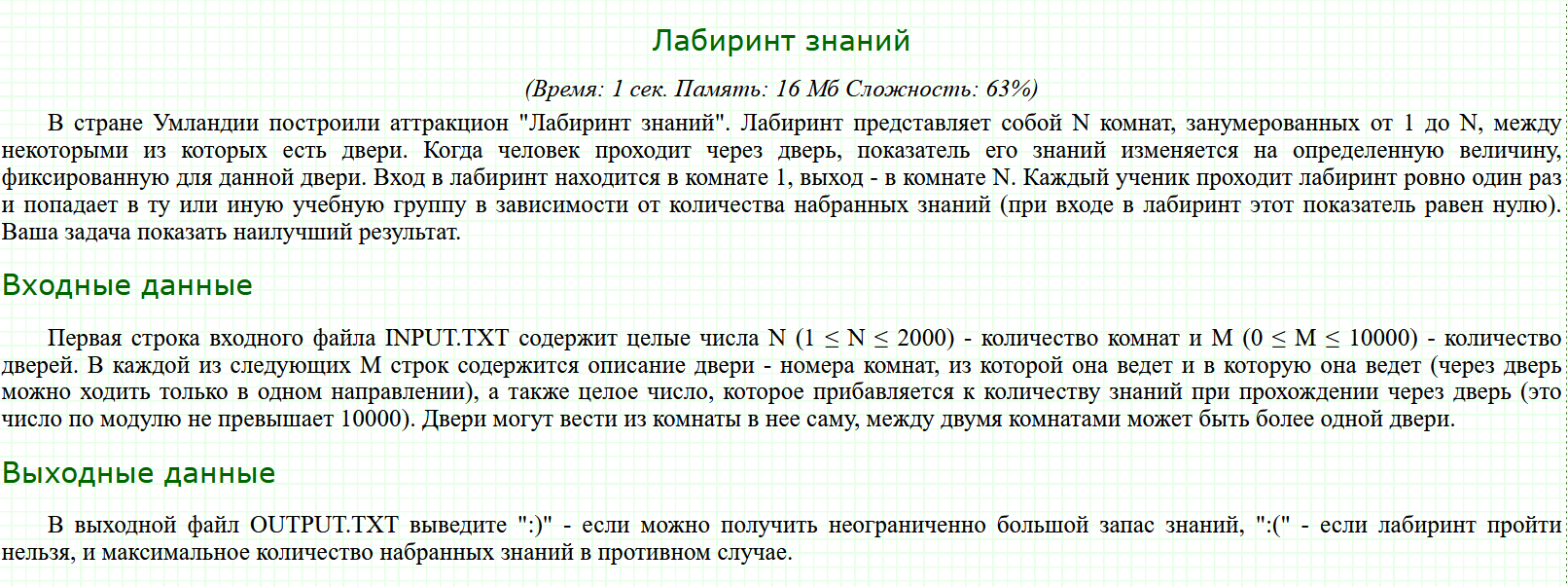
graph = {"start": {"a": 2, "b": 2}, "a": {"c": 2, "end": 2}, "b": {"a": 2},  
 "c": {"b": -1, "end": 2}, "end": {}}  
infinity = float("inf")  
costs = {"a": 2, "b": 2, "c": infinity, "end": infinity}  
parents = {"a": None, "b": None, "c": "a", "end": None}  
processed = []

Код аналогичный

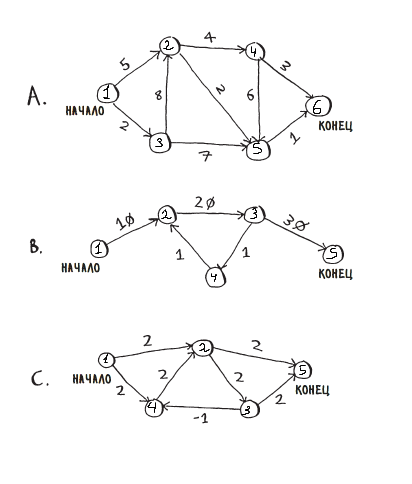
Проблема заключается в том, что одно из ребер имеет отрицательный вес, а алгоритм Дейкстры для того условия не предназначен.

Решением

## Глава 7. Задачи на 5.



from collections import deque  
  
  
n, m = map(int, input().split())  
doors = [list(map(lambda x, y: int(x) + y, input().split(), (-1, -1, 0))) for \_ in range(m)]  
graph = dict([(i, (None,)) for i in range(n)])  
for x, y, z in doors:  
 if graph[x][0] is None:  
 graph[x] = [y]  
 elif y not in graph[x]:  
 graph[x] = graph[x] + [y]  
  
  
def bfs(graph):  
 search\_queue = deque()  
 search\_queue += graph[0]  
 destination = n - 1  
 searched = [0]  
 while search\_queue:  
 point = search\_queue.popleft()  
 if point == destination:  
 return True  
 else:  
 try:  
 search\_queue += list(set(graph[point]) - set(searched))  
 except KeyError:  
 continue  
 searched.append(point)  
 return False  
  
  
def bellman\_ford(n):  
 m\_infinity = -float("inf")  
 dist = [m\_infinity] \* n  
 dist[0] = 0  
 if not bfs(graph):  
 print(":(")  
 return  
 for i in range(n-1):  
 for u, v, w in doors:  
 if dist[u] > m\_infinity and dist[u] + w > dist[v]:  
 dist[v] = dist[u] + w  
 for u, v, w in doors:  
 if dist[u] != m\_infinity and dist[u] + w > dist[v]:  
 print(":)")  
 return  
 print(dist[-1])  
  
  
bellman\_ford(n)



"""  
Условия для графа А  
6 9  
1 2 5  
1 3 2  
2 4 4  
2 5 2  
3 5 7  
3 2 8  
4 6 3  
4 5 6  
5 6 1  
"""  
Вывод: 21  
"""  
Условия для графа Б  
5 5  
1 2 10  
2 3 20  
3 4 1  
3 5 30  
4 2 1  
"""  
Вывод: :)  
"""  
Условия для графа С  
5 7  
1 2 2  
1 4 2  
2 5 2  
2 3 2  
3 5 2  
3 4 -1  
4 2 2  
"""  
Вывод: :)

"""  
Условия для графа А с противоположным знаком  
6 9  
1 2 -5  
1 3 -2  
2 4 -4  
2 5 -2  
3 5 -7  
3 2 -8  
4 6 -3  
4 5 -6  
5 6 -1  
"""  
Вывод: -8  
"""  
Условия для графа Б с противоположным знаком  
5 5  
1 2 -10  
2 3 -20  
3 4 -1  
3 5 -30  
4 2 -1  
"""  
Вывод: -60  
"""  
Условия для графа С с противоположным знаком  
5 7  
1 2 -2  
1 4 -2  
2 5 -2  
2 3 -2  
3 5 -2  
3 4 1  
4 2 -2  
"""

Вывод: -4

# **Глава 8.**

## Глава 8. Задачи на 4.

Реализовать жадный алгоритм для задачи о рюкзаке из главы 8. В качестве критерия выбора предмета рассмотреть стоимость за единицу места в рюкзаке, а не стоимость предмета, как это сделано в книге.

size = 35  
current\_size = 0  
bank = 0  
items = sorted([(3000, 30), (2000, 20), (1500, 15)], key=lambda x: x[0]/x[1])  
for item in items:  
 if item[1] + current\_size <= size:  
 current\_size += item[1]  
 bank += item[0]  
 if current\_size == size:  
 break  
print(bank)

Вывод: 3000

## Глава 8. Задача на 5

Модифицированная задача о покрытии. Список модификаций:

1. Добавить стоимость трансляции с каждой станции
2. Добавить оценку важности каждого штата
3. Критерий выбора станции: сумма важности добавления штатов в покрытие деленная на стоимость трансляции со станции

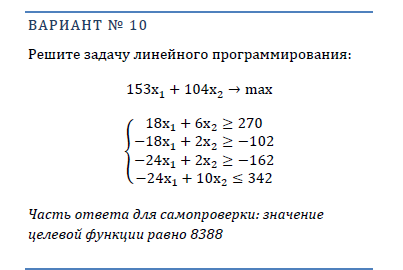
class RadioGaga:  
  
 def \_\_init\_\_(self):  
 self.states\_needed = {"mt", "wa", "or", "id", "nv", "ut", "ca", "az"}  
 self.stations = {  
 'kone': {'id', 'nv', 'ut'}, 'ktwo': {'mt', 'id', 'wa'},  
 'kthree': {'or', 'nv', 'ca'}, 'kfour': {'nv', 'ut'}, 'kfive': {'az', 'ca'}  
 }  
 self.stations\_price = {  
 'kone': 1000, 'ktwo': 2000,  
 'kthree': 1000, 'kfour': 2500, 'kfive': 1700  
 }  
 self.state\_importance = {"mt": 1, "wa": 1, "or": 1, "id": 1, "nv": 1, "ut": 1, "ca": 1, "az": 1}  
 self.final\_stations = set()  
 self.states\_covered = set()  
  
 def add\_station(self, station\_name: str = None, states: list = None, price: int = None):  
 self.stations[station\_name] = states  
 self.stations\_price[station\_name] = price  
 print("Station added")  
  
 def add\_states(self, \*states):  
 self.states\_needed.update(set(states))  
 self.state\_importance.update(dict.fromkeys(states, 1))  
 print("States added")  
  
 def update\_state\_importance(self, state, importance):  
 if state in self.state\_importance:  
 self.state\_importance[state] = importance  
 print(f"{state} importance updated")  
 else:  
 print("No such state in list")  
  
 def \_\_imp\_on\_pr(self, station):  
 if station is None:  
 return -float("inf")  
 importance = 0  
 for state in (self.stations[station] - self.states\_covered):  
 importance += self.state\_importance.get(state, 0)  
 return importance/self.stations\_price[station]

def get\_solution(self):  
 while self.states\_needed:  
 best\_station = None  
 states\_covered = set()  
 for station, states in self.stations.items():  
 covered = self.states\_needed & states  
 if len(covered) < len(states\_covered):  
 best\_station = station  
 states\_covered = covered  
 elif len(covered) == len(states\_covered):  
 if self.\_\_imp\_on\_pr(best\_station) > self.\_\_imp\_on\_pr(station):  
 best\_station = station  
 states\_covered = covered  
 self.states\_covered |= states\_covered  
 self.states\_needed -= states\_covered  
 self.final\_stations.add(best\_station)  
 return self.final\_stations  
  
  
r = RadioGaga()  
print(\*r.get\_solution())

Вывод:

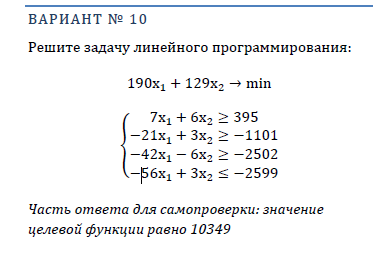
kthree kfive kone ktwo

## Линейное программирование



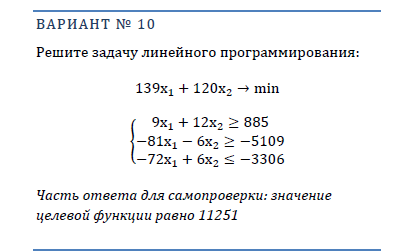
from scipy.optimize import linprog  
  
  
obj = [-153, -104]  
  
lhs\_ineq = [[-18, -6],  
 [18, -2],  
 [24, -2],  
 [-24, 10]]  
  
rhs\_ineq = [-270,  
 102,  
 162,  
 342]  
  
opt = linprog(c=obj, A\_ub=lhs\_ineq, b\_ub=rhs\_ineq,  
 method="highs")  
  
print(-opt.fun)

Вывод: 8388.0



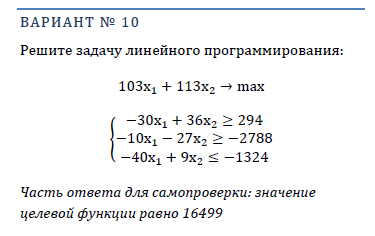
from pulp import LpMaximize, LpProblem, LpStatus, lpSum, LpVariable  
  
model = LpProblem(name="small-problem")  
  
x = LpVariable(name="x")  
y = LpVariable(name="y")  
  
model += (7 \* x + 6 \* y >= 395, "red\_constraint")  
model += (-21 \* x + 3 \* y >= -1101, "blue\_constraint")  
model += (-42 \* x - 6 \* y >= -2502, "yellow\_constraint")  
model += (-56 \* x + 3 \* y <= -2599, "green\_constraint")  
  
obj\_func = 190 \* x + 129 \* y  
model += obj\_func  
status = model.solve()  
print(model.objective.value())

Вывод: 10349.0



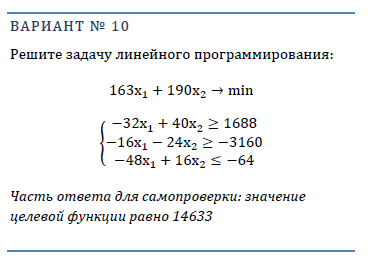
from pulp import LpMaximize, LpProblem, LpStatus, lpSum, LpVariable  
  
model = LpProblem(name="small-problem")  
  
x = LpVariable(name="x")  
y = LpVariable(name="y")  
  
model += (9 \* x + 12 \* y >= 885, "red\_constraint")  
model += (-81 \* x - 6 \* y >= -5109, "blue\_constraint")  
model += (-72 \* x + 6 \* y <= -3306, "yellow\_constraint")  
  
obj\_func = 139 \* x + 120 \* y  
model += obj\_func  
status = model.solve()  
print(model.objective.value())

Вывод: 11251.0



from pulp import LpMaximize, LpProblem, LpStatus, lpSum, LpVariable  
  
model = LpProblem(name="small-problem", sense=LpMaximize)  
  
x = LpVariable(name="x")  
y = LpVariable(name="y")  
  
model += (-30 \* x + 36 \* y >= 294, "red\_constraint")  
model += (-10 \* x - 27 \* y >= -2788, "blue\_constraint")  
model += (-40 \* x + 9 \* y <= -1324, "yellow\_constraint")  
  
obj\_func = 103 \* x + 113 \* y  
model += obj\_func  
status = model.solve()  
print(model.objective.value())

Вывод: 16499.0



from scipy.optimize import linprog  
  
obj = [163, 190]  
  
lhs\_ineq = [[32, -40],  
 [16, 24],  
 [-48, 16]]  
  
rhs\_ineq = [-1688,  
 3160,  
 -64]  
  
opt = linprog(c=obj, A\_ub=lhs\_ineq, b\_ub=rhs\_ineq,  
 method="highs")  
  
print(opt.fun)

Вывод: 14633.0