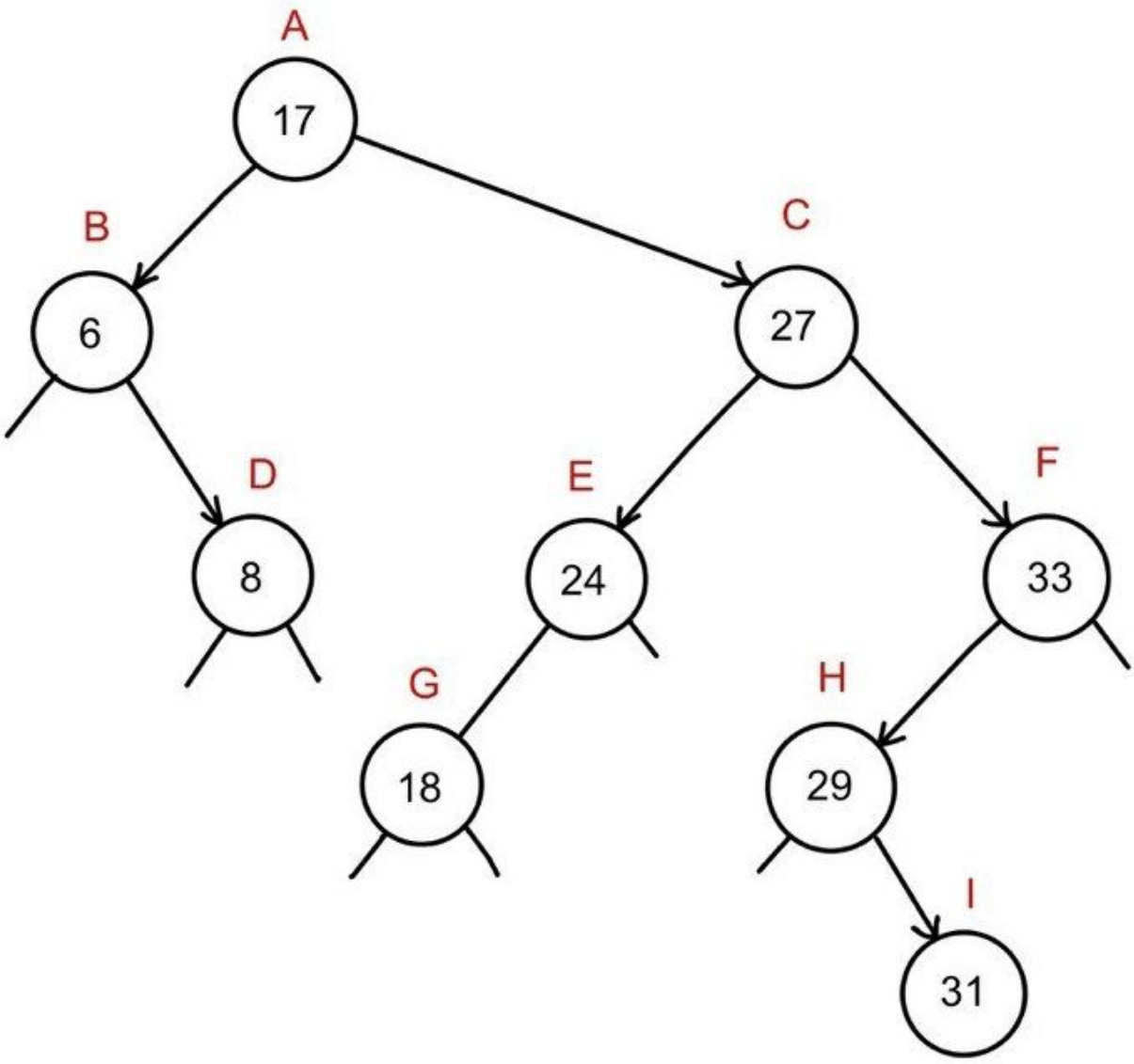
General test

1. На малюнку нижче зображено бінарне дерево пошуку. Визначте порядок обходу дерева при пошуку floor(28), і вузол, що містить результат. Також визначте складність операції floor у найгіршому та середньому випадках.
   Операція floor повертає найбільший ключ, що менше або дорівнює заданого ключа. Наприклад, якщо у дереві є ключі {1, 3, 5, 7}, то floor(6)=5.



Відповіді:

1) Складність у середньому випадку – O(log(n))
2) Складність у найгіршому випадку – O(n)
3) Порядок обходу дерева при пошуку floor(28) – Порядок: A, C, F, H
4) Вузол, що містить результат – Вузол: C

2. Визначте оптимальну структуру даних для зазначеного сценарію використання, а також загальну (для всіх дій у сценарії) асимптотичну складність. Сценарій: створити структуру даних з N (N = 1,000,000) елементів, потім виконати M (M = 1,000,000) операцій, серед яких: 25% вставок, 65% пошуків мінімального значення, 8% видалень мінімального значення, 1% пошуків максимального значення, 1% видалень максимального значення.
   При аналізі складності операцій вважайте, що у структурі даних знаходиться N елементів на момент виконання операції. Оптимальною є та структура даних, що дозволяє виконати всі дії у сценарії за мінімальний час.
   Відповідь: Оптимальна структура даних – AVL-дерево. Загальна складність: O((N+M)*log(N))
3. На малюнку нижче подано код хеш-функції. Для хеш-таблиці, що базується на методі ланцюжків, визначте середню кількість ключів, що припадає на одну комірку, якщо до хеш-таблиці з load factor = 75% було додано ключі з таких діапазонів: 1) з діапазону [1000000;2000000) – 1 мільйон ключів (тобто всі числа з діапазону без повторень); 2) з діапазону [2000000;3000000) – 100 тисяч випадкових ключів.
   Для другого діапазону під "випадковими" ключами мається на увазі, що усі значення з діапазону рівноймовірні. Це означає, що у кожному достатньо великому проміжку буде однакова (у контексті цієї задачі) кількість ключів. Наприклад, у діапазоні [2000000;2100000) буде 10 тисяч ключів, і у діапазоні [2100000;2200000) – теж 10 тисяч ключів.

```
def hash(number: int) -> int:
    return int(str(number)[:3])
```

   Відповідь: 5500
4. Оберіть правильне твердження щодо знаходження найменшого елементу у максимальній купі з N елементів.
   Відповідь: Найменший елемент можна знайти за O(N) у найгіршому випадку, він знаходиться у листках дерева
5. На малюнку нижче подано код реалізації методу видалення довільного елементу з мінімальної купи (min heap), реалізованої на базі масиву. Якою є асимптотична складність роботи цього методу в найгіршому випадку, якщо в купі міститься N елементів?
   Це фрагмент реалізації класу Heap. Поле arr – це list у Python.

```
def delete(self, element):
    if self.size() == 0:
        return None
    element_index = self.arr.index(element)
    self.arr[element_index], self.arr[-1] = self.arr[-1], self.arr[element_index]
    self.arr.pop()
    if self.size() > 0:
        self.sift_down(0)
```

Відповідь: O(n)

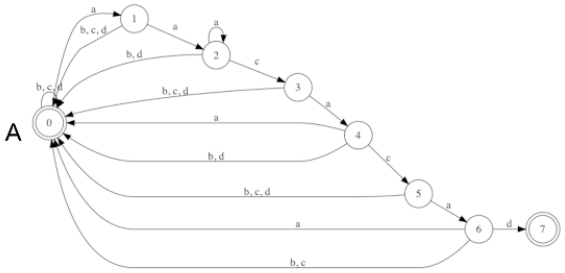
6. Розглянемо хеш-таблицю, що заснована на методі ланцюжків. Початкова ємність внутрішнього масиву – 9 комірок. Load factor = 60%. Кожен раз, коли наповненість масиву досягає load factor, його розмір збільшується на 9 елементів (операція resize). Установіть правильні відповідності.

   Відповіді:

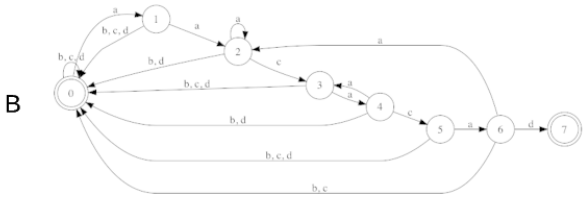
   1) Асимптотична складність додавання N елементів у середньому випадку – O(N^2)
   2) Асимптотична складність додавання одного елементу у найгіршому випадку – O(N)
   3) Асимптотична складність додавання N елементів у найгіршому випадку – O(N^2)
   4) Асимптотична складність додавання одного елементу у найкращому випадку – O(1)

7. Дано рядок 'aacacad'. Оберіть зображення, яке показує правильний скінченний автомат, побудований алгоритмом Кнута — Морріса — Пратта (КМР).

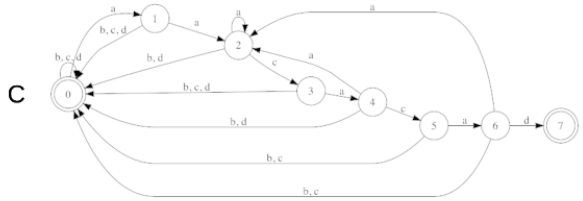
A



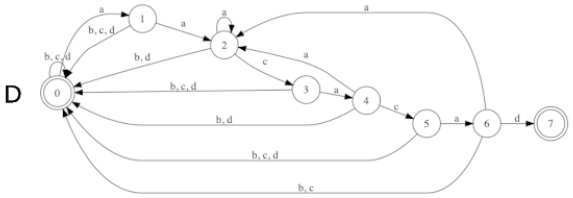
B



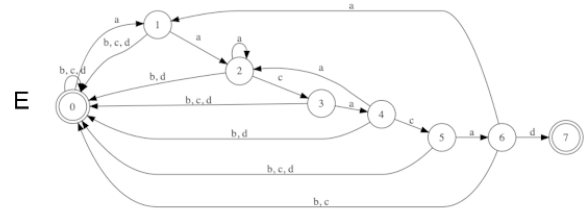
C



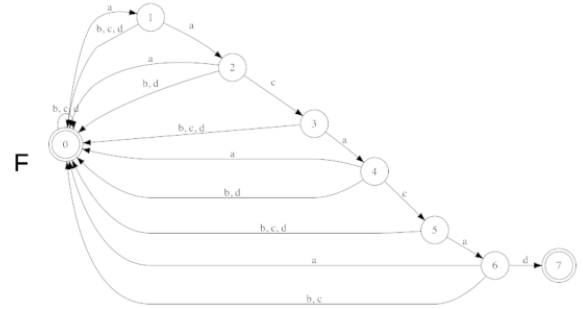
D



E

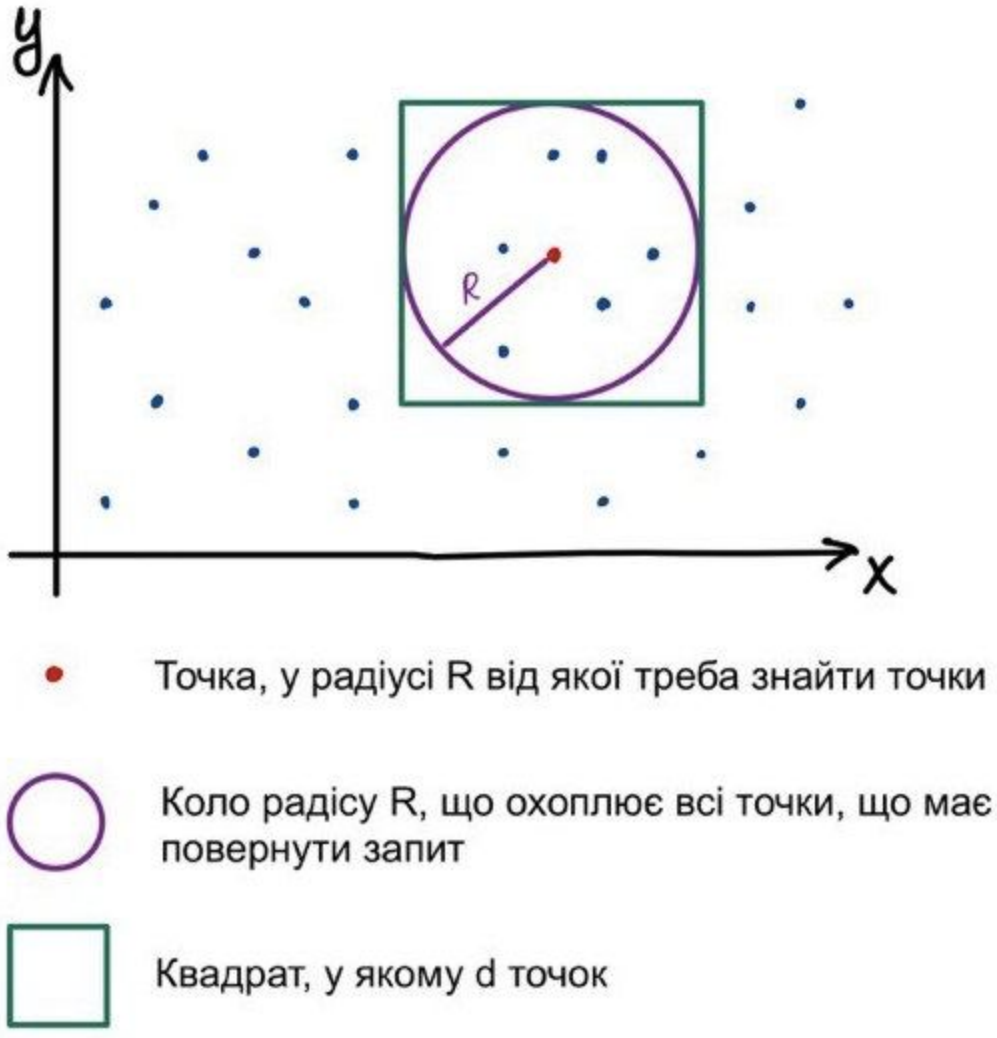


F



Відповідь: D

8. Оберіть усі правильні твердження.
   Позначення: KMP – алгоритм Кнута-Морріса-Пратта; RK – алгоритм Рабіна-Карпа; N – довжина рядку (тексту), у якому виконується пошук патерну; R – розмір абетки; M – довжина шуканого патерну; DFA – детермінований скінченний автомат.
   Відповіді:
   1) Приблизний алгоритм RK має складність O(N) у найгіршому випадку
   2) Найгірший випадок (відносно аналізу обчислювальної складності) для алгоритму brute force (грубої сили) – коли вхідний текст і патерн містять символи, що повторюються
9. Дано: множина з N точок у двовимірному просторі. Потрібно реалізувати запити на знаходження точок у певному радіусі R від заданої точки (x, y). Якою буде складність виконання M запитів для: 1) алгоритму повного перебору; 2) запитів до індексу, заснованого на AVL дереві? Вважайте, що у квадраті з вершинами (x-r, y+r), (x+r, y+r), (x-r, y-r), (x+r, y-r) у середньому знаходиться d точок. Наведений нижче малюнок ілюструє цю логіку.

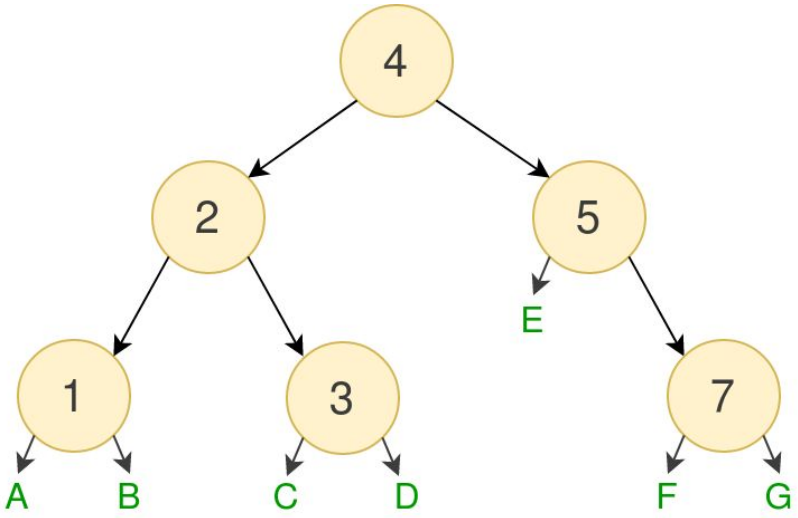


   Відповіді:

1) Алгоритм повного перебору – O(M*N)
2) Індекс, заснований на AVL дереві – O(N*log(N) + M*(d+log(N)))

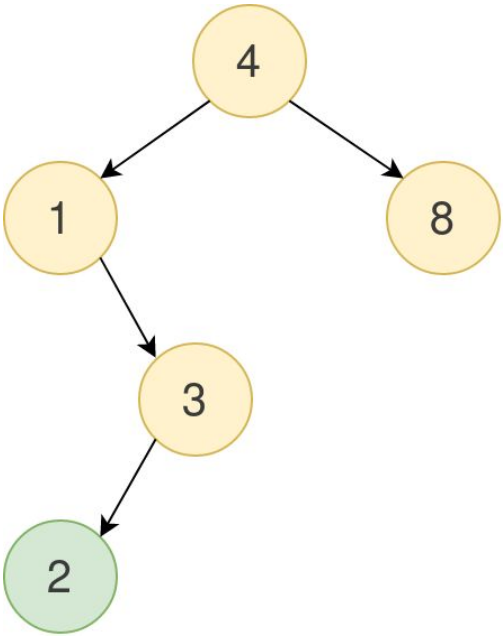
---

1. (heap) Маємо бінарну мінімальну купу з N елементів і хочемо вставити в неї ще N елементів. Яка можлива найбільш ефективна складність такої операції? **Відповідь: O(N)**
2. (heap) Маємо мінімальну бінарну купу з N елементів. Задача — перетворити її в максимальну бінарну купу. Яка можлива найбільш ефективна складність такої операції? **Відповідь: O(N)**
3. (BST/AVL) Яка складність вставки в бінарне дерево пошуку в найгіршому випадку? **Відповідь: O(N)**
4. (BST/AVL) Яка складність пошуку в AVL-дереві в найгіршому випадку? **Відповідь: O(log(N))**
5. (BST/AVL) Якою є складність одного повороту в AVL-дереві в залежності від кількості елементів в ньому? **Відповідь: O(1)**
6. (BST/AVL) Дано бінарне дерево пошуку. В нього необхідно вставити елемент “6”. В якому місці він буде вставлений?



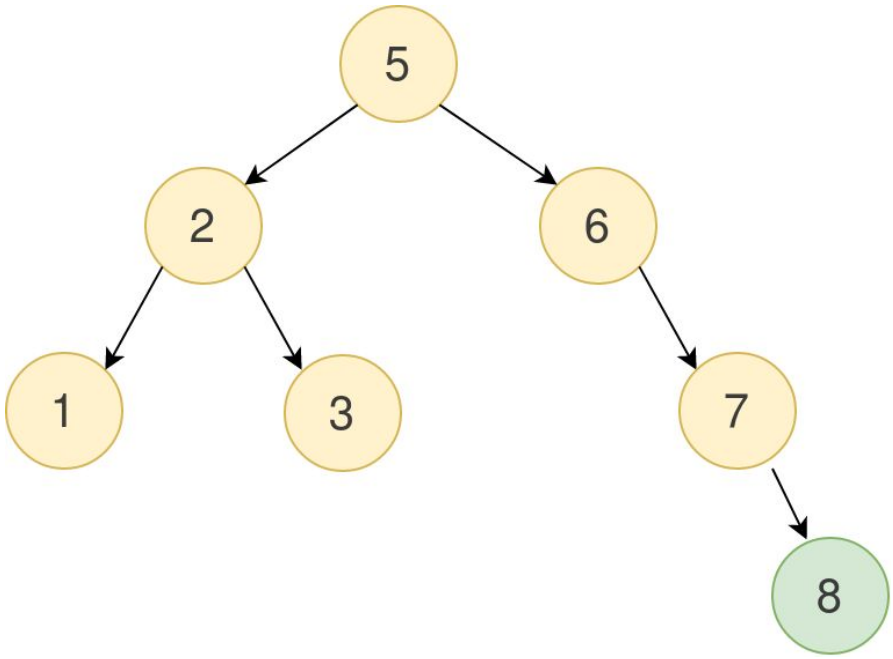
Відповідь: F

7. (BST/AVL) Дано AVL-дерево. В нього щойно був вставлений елемент “2”. При цьому, балансування ще не було виконане. Чи потрібно робити балансування і якщо так, то яким чином?



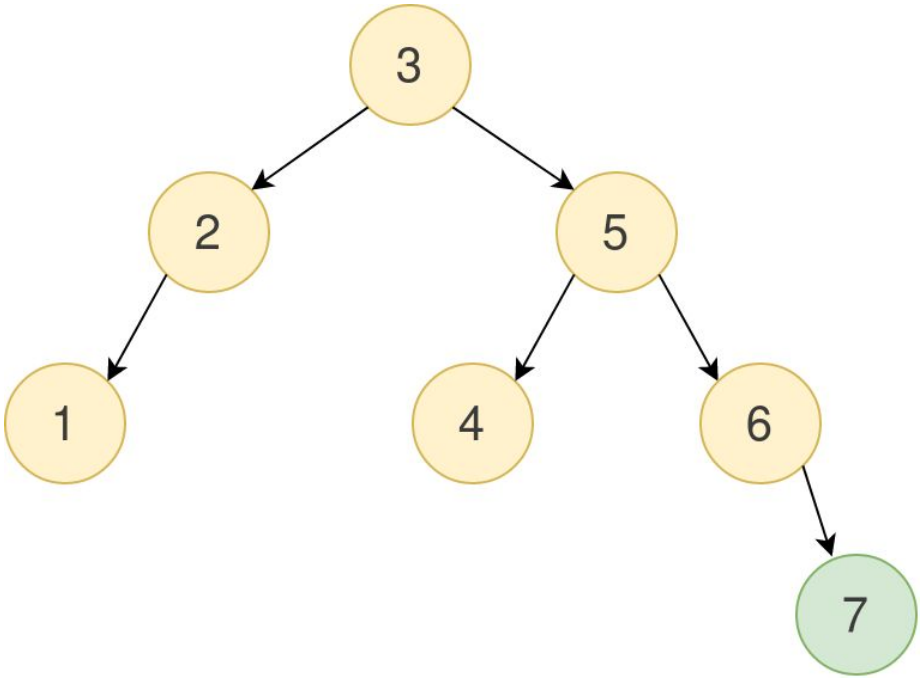
Відповідь: Необхідно зробити правий поворот вузла "3", а потім лівий поворот вузла "1"

8. (BST/AVL) Дано AVL-дерево. В нього щойно був вставлений елемент “8”. При цьому, балансування ще не було виконане. Чи потрібно робити балансування і якщо так, то яким чином?



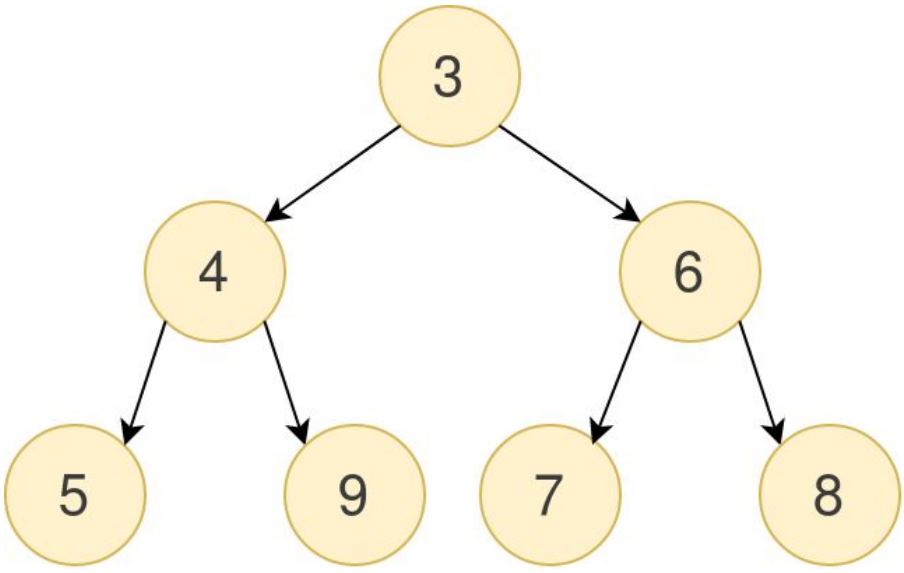
Відповідь: Необхідно зробити лівий поворот вузла "6"

9. (BST/AVL) Дано AVL-дерево. В нього щойно був вставлений елемент “7”. При цьому, балансування ще не було виконане. Чи потрібно робити балансування і якщо так, то яким чином?



Відповідь: Балансування робити непотрібно, дерево вже збалансоване

10. (heap) На зображенні показано мінімальну купу (min heap). Яким чином її елементи були б розміщені у відповідному масиві?



Відповідь: [3, 4, 6, 5, 9, 7, 8]

11. (Prefix trees) Пошук ключа довжини K в префіксному дереві висотою L відбувається за: O(K*log(L)), **O(K)**, O(L), O(K*L)

12. (DP)Parent Pointers в динамічному програмуванні дозволяють: **Дізнатись, яка послідовність рішень підзадачі привела до рішення оригінальної задачі**. Порахувати складність роботи програми. Зберегти рішення підзадач для перевикористання на наступних ітераціях алгоритму. Побудувати Bottom-up варіант рішення задачі.
13. (Edit distance) Яка відстань Левенштейна між словами “cat” та “snap”? Вартості всіх операцій по 1. 1, 2, **3**, 4, 5
14. (Edit distance)Яка відстань Левенштейна між словами “cat” та “snap”? Вартість вставки та видалення по 2, заміни: 1. Замінювати на пустий символ не дозволяється. 1, 2, 3, **4**, 5
15. (Edit distance)Яке число буде в клітинці з “?” при обчисленні відстані Левенштейна?

	“”	“w”	“a”	“r”
“”	0	1	2	3
“w”	1	0	1	
“i”	2	1	?	
“n”	3			

0, **1**, 2, 3

16. (Edit distance)Яке число буде в клітинці з “?” при обчисленні відстані Левенштейна?

	“”	“m”	“a”	“n”
“”	0	1	2	3
“c”	1	1	2	3
“a”	2	2	1	
“v”	3	3	?	

0, 1, **2**, 3

17. (Edit distance)Яке число буде в клітинці з “?” при обчисленні відстані Левенштейна, при цьому **вартість заміни: 3, вставки та видалення: 1**?

	“”	“m”	“a”	“n”
“”	0	1	2	3
“c”	1	1	2	3
“a”	2	2	1	?
“v”	3	3		

0, 1, **2**, 3

18. (LSH)Яка складність етапу підготовки K структур даних для множини з Z точок розмірності t при виконанні LSH? Операція хешування однієї точки виконується за O(1). **O(Z*K)**, O(Z*t), O(Z*K*t), O(Z)
19. (LSH)Припустимо, що при виконанні LSH розподіл точок по хеш-таблицях виявився таким:

HashTable1			HashTable2		HashTable3		HashTable4	
A	B	C	D	F	B	A	D	E
E	D	Z	E	H	D	F	I	H
F	G	X	B	A	I	G	B	F
H	I	Y	I	C	H	B	A	C

Який поріг потрібно обрати для того, щоб точки A, B та C вважались сусідніми? **0.5**, 0.66, 0.75, 0.9

Lab2 test

1. На малюнку нижче наведено код розв'язку задачі розміну монет. Задача формулюється так: маючи множину з M номіналів монет S = {S1, S2, ..., Sm} і нескінченну кількість монет кожного номіналу, потрібно знайти кількість способів, якими можна розміняти одну монету номіналу N. Для наведеного коду визначте асимптотичну складність по часу та пам'яті, а також знайдіть помилку. Наприклад, для N = 4 і S = {1, 2, 3} є чотири способи розміняти: {1, 1, 1, 1}, {1, 1, 2}, {2, 2}, {1, 3}. Інший приклад: для N = 10 і S = {2, 5, 3, 6}, розв'язок має виводити 5: {2, 2, 2, 2, 2}, {2, 2, 3, 3}, {2, 2, 6}, {2, 3, 5} and {5, 5}.

```python
def count(S, n):
    m = len(S)
    table = [[0 for _ in range(m)] for _ in range(n + 1)]

    for i in range(m):
        table[0][i] = 1

    for i in range(1, n + 1):
        for j in range(m):
            x = table[i - S[j]][j] if j - S[i] >= 0 else 0
            y = table[i][j - 1] if j >= 1 else 0
            table[i][j] = x + y

    return table[n][m - 1]
```

Відповідь: O(M*N) по часу; O(M*N) по пам'яті; помилка в рядку №10 – має бути 'x = table[i - S[j]][j] if i - S[j] >= 0 else 0'

2. Дано: N файлів, у кожному з яких міститься M рядків формату "<ключ> <рядок>". Потрібно об’єднати файли так, щоб у результуючому файлі були записи "<ключ> <рядок>", відсортовані по ключу, і тільки ті записи, де значення ключа більше або дорівнює заданому значенню (K). Визначте асимптотичну обчислювальну складність (у найгіршому випадку) оптимального розв’язку для двох випадків: 1) кожен вхідний файл (незалежно від інших) відсортований по ключу; 2) вхідні файли невпорядковані.
   Один і той же ключ може зустрічатися в декількох файлах. У результуючому файлі порядок рядків з однаковими значеннями ключа не має значення. Приклад розв'язку для другого випадку (файли невпорядковані) зображено на малюнку.

Параметри: N = 3, M = 5, K = 4

Файл 1	Файл 2	Файл 3	Результуючий файл
3 a **4 b** 2 c **5 d** **7 e**	1 f 3 g 2 h **9 i** **4 j**	2 k **5 l** 3 m **6 n** 1 o	4 b 4 j 5 d 5 l 6 n 7 e 9 i

Відповіді:

1) Кожен вхідний файл (незалежно від інших) відсортований по ключу – M*N*log(N)

2) Вхідні файли невпорядковані – M*N*(log(M)+log(N))

```
def whatIsIt(arr, arr_size):
    a = arr[1] - arr[0]
    b = arr[0]

    for i in range( 1, arr_size ):
        if (arr[i] - b > a):
            a = arr[i] - b

        if (arr[i] < b):
            b = arr[i]
    return a

arr = [1, 2, 6, 80, 100]
size = len(arr)
print ("Шукане значення = ", whatIsIt(arr, size))
```

1. Що обчислює функція **whatIsIt**? **Відповідь: Максимальну різницю між елементами масиву**
2. Задано масив чисел N, де кожне елемент в масиві — ціле число більше або рівне нуля. Яка можлива найбільш ефективна складність алгоритму, котрий перенесе всі нулі в кінець масиву? Приклад: алгоритм перетворить масив [1, 0, 2, 6, 0, 4] у [1, 2, 6, 4, 0, 0]. **Відповідь: O(N)**
3. Якою буде складність пошуку найменшого ключа (в лексикографічному порядку) в префіксному дереві в найгіршому випадку, якщо в ньому знаходиться N ключів і всі вони довжиною не більше за K символів. При цьому, всі ключі містять тільки маленькі букви латинського алфавіту.
    a. **O(k)**
    b. O(n)
    c. O(log(n))
    d. O(log(k))
    e. O(n*log(n))
    f. O(n*log(k))
    g. O(n + k)
    h. O(k*log(n))
4. Якою стандартною функцією python (без використання import) можна отримати код символа в ASCII-таблиці
    a. **ord()**
    b. chr()
    c. int()
    d. char()
    e. len()

    f. code()
    g. str()
    h. ascii()
5. Мемоізація дозволяє перевикористати рішення підзадачі за: O(n), **O(1)**, O(log(n)), O(n*log(n))
6. Яким чином обчислюється складність при вирішенні задачі через динамічне програмування: **O(кількість_підзадач * час_вирішення_підзадачі)**, O(час_вирішення_підзадачі), O(кількість_підзадач), O(кількість_підзадач + час_вирішення_підзадачі)