Київський національний університет імені Тараса Шевченка

КРЕНЕВИЧ А.П.
КЛЕВЦОВСЬКИЙ А.В.

Методичні вказівки до лабораторних занять із дисципліни

«Алгоритми і структури даних»

для студентів механіко-математичного факультету

Київ – 2020

УДК 519.942+550

Рецензенти:
доктор фіз.-мат. наук, професор
доктор фіз.-мат. наук,

Рекомендовано до друку вченою радою механіко-математичного факультету
(протокол № __ від __ _____202__ року)

Креневич А.П.

Методичні вказівки до лабораторних занять із дисципліни «Алгоритми і структури даних» для студентів механіко-математичного факультету / А. П. Креневич, А. В. Клевцовський. – К.: ВПЦ "Київський Університет", 2020. – ___ с.

Посібник містить перелік завдань для самостійної роботи з дисципліни «Алгоритми і структури даних», що викладається студентам механіко-математичного факультету Київського національного університету імені Тараса Шевченка. Він містить завдання для засвоєння основних понять цього курсу, таких як рекурентні співвідношення, рекурсія, аналіз складності алгоритмів, лінійні рекурсивні структури даних, графи, дерева, динамічне програмування тощо.

Для студентів механіко-математичного факультету та викладачів, які проводять заняття з курсу «Алгоритми і структури даних».

ЗМІСТ

ВСТУП

ВИМОГИ ДО ОФОРМЛЕННЯ ЛАБОРАТОРНИХ РОБІТ

1. Основна ціль лабораторної роботи довести викладачу, що студент засвоїв матеріал на достатньому рівні.
2. Кожна лабораторна роботи містить
    a. перелік контрольних запитань, що стосуються теоретичного матеріалу;
    b. варіанти індивідуальних/групових завдань лабораторних робіт для практичного виконання;
    c. посилання на допоміжну літературу для закріплення теоретичного матеріалу і стане в нагоді під час виконання практичних завдань.
3. На вибір студентів надано перелік практичних завдань по кожній темі, серед яких вони мають можливість самостійно вибрати бажаний варіант. Обраний номер завдання студенти записують у спеціальний хмарний документ, наданий викладачем, ціль якого рівномірно розподілити завдання між студентами групи. Кілька студентів однієї групи, що можуть обрати один варіант не може перевищувати 2, у випадку самостійної роботи, та 3-х, у випадку роботи у команді.
4. Кожна лабораторна робота оформлюється у паперовому та електронному вигляді.
5. Паперове оформлення здійснюється за допомогою текстового процесору Word (чи аналогічного) або редактора презентацій PowerPoint (чи аналогічного) та має містити такі частини:
    a. Титульний аркуш, що містить назву навчальної дисципліни, номер лабораторної роботи, номер варіанту, ПІБ студента (студентів, якщо спільна лабораторна робота), що виконав роботу, спеціальність, курс, номер групи.
    b. Умову задачі. Якщо задача береться з електронних джерел, то вказати посилання на задачу. Умова задачі має містити приклади вхідних та вихідних даних.
    c. Аналіз задачі та опис алгоритму, яким пропонується розв’язувати задачу.
    d. Приклад роботи алгоритму на модельному прикладі (взятому з умови задачі або придуманому самостійно).
    e. Програмну реалізацію (основні моменти).
    f. Висновки: якщо робота програми перевірялася за допомогою електронних систем, то обов’язково вказати посилання на систему перевірки разом з задачею (наприклад, https://www.e-olymp.com/uk/problems/3966), логін (логіни)

виконавців лабораторної роботи та результат перевірки (наприклад, 100%). Якщо результат не 100%, то обов'язково зазначити у чому проблема (не пройшов, по часу, помилки виконання, тощо).

6. Електронна частина, складається з
    a. Файлу (файлів), з яких друкується паперовий примірник лабораторної роботи.
    b. Файлів вихідного коду програми, що розв'язує поставлену задачу.
    c. Файлів вхідних даних (за необхідності) – текстові файли, що місять вхідні дані задачі. Вони мають бути названі згідно з правилом – «input.txt» (якщо один файл) та «input01.txt», «input02.txt» і т.д. – якщо кілька файлів вхідних даних.
7. Електронний варіант надсилається на електронну адресу викладача, що викладає практичні заняття не пізніше ніж напередодні лабораторного заняття, на якому студент має її захищати.
8. Захист лабораторної роботи здійснюється студентом (студентами) одним з двох способів:
    a. Індивідуальна співбесіда з викладачем, протягом якої викладач ставить запитання по теоретичному матеріалу теми лабораторної роботи (з переліку контрольних запитань), вислуховує ідею алгоритму, що розв'язує поставлену задачу, знайомиться з деталями реалізації алгоритму та перевіряє коректність розв'язку.
    b. Презентація розв'язаної задачі з усіма деталями реалізації для всіх студентів навчальної групи. У цьому випадку роздрукований варіант не вимагається.

ЛАБОРАТОРНА РОБОТА 1. Складність алгоритмів. Час виконання.

Контрольні запитання

1.1. Наведіть означення алгоритму.
1.2. Що таке комп'ютерна програма? Що таке реалізація алгоритму?
1.3. У чому полягає аналіз алгоритму?
1.4. Що таке елементарна операція? Наведіть кілька прикладів.
1.5. Що таке час виконання програми?
1.6. Що означає фраза: час виконання програми сталий/логарифмічний/лінійний/поліноміальний/експонеційний?
1.7. Наведіть означення таких понять як найкращий/найгірший час виконання програми. Що таке час виконання програми в середньому.

Завдання для аудиторної та самостійної роботи

1.1. Визначте час виконання фрагментів програм заданих нижче

a)
```
1   i = 0
2   while i < n:
3       k += 1
4       i += 1
```

b)
```
1   i = 1
2   while i < n:
3       i = i * 2
```

c)
```
1   i = 0
2   while i < n:
3       if i % 2 == 0:
4           k += 1
5       i += 1
```

d)
```
1   i = 0
2   while i < n:
3       j = n
4       while j > 0:
5           k += 1
6           j -= 1
7       i += 1
```

e)
```
1   i = 0
2   while i < n:
3       j = i
4       while j < n:
5           k += 1
6           j += 1
7       i += 1
```

f)
```
1   i = 0
2   while i < n:
3       j = n
4       while j != 0:
5           k += 1
6           j //= 3
7       i += 1
```

1.2. Визначте час виконання програми у явному вигляді, якщо для нього відоме рекурентне співвідношення

a) $T(n) = \begin{cases} 1, & n = 0; \\ T(n-1) + 1, & n \geq 1. \end{cases}$

b) $T(n) = \begin{cases} 1, & n = 0; \\ 2T(n-1) + 1, & n \geq 1. \end{cases}$

c) $T(n) = \begin{cases} 1, & n = 1; \\ T([n/a]) + 1, & n \geq 2. \end{cases}$

d) $T(n) = \begin{cases} 1, & n = 1; \\ 2T([n/a]) + 1, & n \geq 2. \end{cases}$

Завдання для самостійної роботи

1.3. Визначте час виконання фрагментів програм заданих нижче

a)
```
k += 1
i = n
while i > 0:
    i -= 1
```

b)
```
i = n
while i > 1:
    k += 1
    i //= 2
```

c)
```
i = 0
while i < n:
    j = 0
    while j < n:
        k += 1
        j += 2
    i += 2
```

d)
```
i = 0
while i < n:
    j = 0
    while j = i * i:
        k += 1
        j += 1
    i += 1
```

e)
```
i = 1
while i < n:
    j = 1
    while j < n:
        k += 1
        j *= 2
    i *= 2
```

f)
```
i = 1
while i < n:
    j = i
    while j < n:
        k += 1
        j *= 2
    i *= 2
```

1.4. Визначте час виконання програми у явному вигляді, якщо для нього відоме рекурентне співвідношення

a) $T(n) = \begin{cases} 1, & n \leq a, a > 0; \\ T(n-a) + 1, & n \geq a. \end{cases}$

b) $T(n) = \begin{cases} 1, & n = 0; \\ 2T(n-1) + n, & n \geq 1. \end{cases}$

c) $T(n) = \begin{cases} 1, & n = 1; \\ 2T([n/a]) + n, & n \geq 2. \end{cases}$

d) $T(n) = \begin{cases} 1, & n = 0; \\ aT([n/a]) + n^2, & n \geq 1, a \geq 2. \end{cases}$

ЛАБОРАТОРНА РОБОТА 2. Складність алгоритмів. Асимптотична оцінка складності алгоритмів.

Контрольні запитання

2.1. Наведіть означення О-оцінки: що означає $f = O(g)$?
2.2. Наведіть означення Омега-оцінки: що означає $f = \Omega(g)$?
2.3. Наведіть означення Тета-оцінки: що означає $f = \Theta(g)$?
2.4. Наведіть основні правила визначення асимптотичної поведінки часу виконання.

Завдання для аудиторної роботи

2.1. Нехай $f(n) = 3n^2 - n + 4$. Користуючись означенням покажіть що

a) $f(n) = O(n^2)$, b) $f(n) = \Omega(n^2)$.

2.2. Для кожної пари функцій $f(n)$ та $g(n)$, зазначених у таблиці визначте яке із співвідношень має місце $f(n) = O\big(g(n)\big)$ чи $g(n) = O\big(f(n)\big)$

$f(n)$	$g(n)$
$10n$	$n^2 - 10n$
n^3	$n^2 \log n$
$n \log n$	$n + \log n$
$\log n$	$\sqrt[k]{n}$
$\ln n$	$\log n$
$\log(n+1)$	$\log n$
$\log \log n$	$\log n$
2^n	10^n
n^m	m^n
$\cos(n\pi/2)$	$\sin(n\pi/2)$
n^2	$(n \cos n)^2$

2.3. Визначте асимптотичну оцінку виконання функції у термінах O– «великого» для функції:

```
def f(n):
    k = 0
    i = n - 1
    while i != 0:
       k += 1.0 / i
       i = i / 2
    return k
```

Що є результатом виконання наведеної функції для заданого натурального числа n?

2.4. Визначте асимптотичну оцінку виконання функції у термінах O–«великого» для функції:

```
def f(n):
    i = n - 1
    while i != 0:
        j = 0
        while j < n:
            j += 1
        i = i / 2
```

2.5. Знайдіть асимптотичний час виконання програми у явному вигляді, якщо для нього відоме рекурентне співвідношення

a) $T(n) = \begin{cases} O(1), & n = 0; \\ 2T(n-1) + O(1), & n \geq 1. \end{cases}$

b) $T(n) = \begin{cases} O(1), & n = 0; \\ 2T(n-1) + O(n), & n \geq 1. \end{cases}$

c) $T(n) = \begin{cases} O(1), & n = 0; \\ 2T([n/2]) + O(1), & n \geq 1. \end{cases}$

d) $T(n) = \begin{cases} O(1), & n = 0; \\ 2T([n/2]) + O(n), & n \geq 1. \end{cases}$

2.6. Чи можна описати алгоритм для кожної з задач підрахунку суми наведених нижче, асимптотична складність яких буде $O(1)$? Якщо так, то наведіть фрагмент такої програми.

a) $\sum_{i=0}^{n} i$ b) $\sum_{i=0}^{n} a^i$ c) $\sum_{i=0}^{\infty} a^i, |a| \leq 1$

Завдання для самостійної роботи

2.7. Нехай $f(n) = 3n^2 - n + 4$ та $g(n) = n \log n + 5$. Покажіть що

$$f(n) + g(n) = O(n^2).$$

2.8. Нехай $f(n) = \sqrt{n}$ та $g(n) = \log n$. Покажіть що

$$f(n) + g(n) = O(\sqrt{n}).$$

2.9. Припустимо, що n, m та k невід'ємні цілі числа і методи e, f, g та h мають такі характеристики:

- Час виконання у найгіршому випадку методу є `e(n, m, k)` є $O(1)$ і повертає значення з проміжку від 1 до $(n + m + k)$.
- Час виконання у найгіршому випадку методу є `f(n, m, k)` є

$O(n+m)$.

- Час виконання у найгіршому випадку методу є g(n, m, k) є $O(m+k)$.
- Час виконання у найгіршому випадку методу є h(n, m, k) є $O(n+k)$.

Визначте асимптотичні оцінки виконання програм у найгіршому випадку в термінах O–«великого» для фрагментів програм зазначених нижче:

a)
```
1   f(n, 10, 0)
2   g(n, m, k)
3   h(n, m, 1000000)
```

b)
```
1   for i in range(e(n, 10, 100)):
2      f(n, 10, 0);
```

c)
```
1   for i in range(n):
2      f(n, m, k);
```

d)
```
1   for i in range(n):
2      for j in range(i, n):
3          f(n, m, k);
```

2.10. Визначте асимптотичну оцінку виконання функції у найгіршому випадку в термінах O–«великого» для функції:

```
def f(n):
    sum = 0
    for i in range(1, n + 1):
        sum = sum + i
    return sum
```

Що є результатом виконання наведеної функції для заданого натурального числа n? Чи можна оптимізувати цю функцію, покращивши її асимптотичну оцінку?

2.11. Нехай f(n) функція визначена у вправі 2.10. Розглянемо функцію

```
def g(n):
    sum = 0
    for i in range(1, n + 1):
        sum = sum + i + f(i)
    return sum
```

Визначте асимптотичну оцінку виконання функції g(n) у найгіршому випадку в термінах O–«великого». Що є результатом виконання функції g(n) для заданого натурального числа n. Чи можна оптимізувати цю функцію, покращивши її асимптотичну оцінку?

2.12. Нехай f(n) функція визначена у вправі 2.10, а функція g(n) – у вправі 2.11. Розглянемо функцію

```
def h(n):
    return f(n) + g(n)
```

Визначте асимптотичну оцінку виконання функції h(n) у найгіршому випадку в термінах O–«великого». Що є результатом її виконання для заданого натурального числа n. Чи можна оптимізувати цю функцію, покращивши її асимптотичну оцінку?

2.13. Знайдіть асимптотичний час виконання програми у явному вигляді, якщо для нього відоме рекурентне співвідношення

a) $T(n) = \begin{cases} O(1), & n = 0; \\ T(n-1) + O(1), & n \geq 1. \end{cases}$

b) $T(n) = \begin{cases} O(1), & n = a, \ a > 1; \\ T(n-a) + O(1), & n \geq a + 1. \end{cases}$

c) $T(n) = \begin{cases} O(1), & n = 0; \\ aT(n-1) + O(1), & n \geq 1, a > 1. \end{cases}$

d) $T(n) = \begin{cases} O(1), & n = a, \ a > 1; \\ aT(n-a) + O(1), & n \geq a + 1. \end{cases}$

e) $T(n) = \begin{cases} O(1), & n = 0; \\ aT(n-1) + O(n), & n \geq 1, a > 1. \end{cases}$

f) $T(n) = \begin{cases} O(1), & n = 0; \\ aT(n-1) + O(n), & n \geq 1, a > 1. \end{cases}$

g) $T(n) = \begin{cases} O(1), & n = 0; \\ aT([n/\mathrm{a}]) + O(1), & n \geq 1, a \geq 2. \end{cases}$

h) $T(n) = \begin{cases} O(1), & n = 0; \\ aT([n/\mathrm{a}]) + O(n), & n \geq 1, a \geq 2. \end{cases}$

2.14. Доведіть співвідношення

a) $\sum_{i=0}^{n} i = O(n)$ b) $\sum_{i=0}^{n} i^2 = O(n^2)$ c) $\sum_{i=0}^{n} i^3 = O(n^4)$

d) $\sum_{i=0}^{n} a^i = O(n)$ e) $\prod_{i=1}^{n} \frac{1}{1+i} = O(n)$ f) $\prod_{i=1}^{n} \frac{1}{1+i^2} = O(n^2)$

g) $\prod_{i=1}^{n} \frac{1}{1+i!} = O(n)$ h) $\prod_{i=1}^{n} \frac{1}{1+i^m} = O(nm)$ i) $\prod_{i=1}^{n} \frac{1}{1+i^i} = O(n^2)$

Зауваження. Для доведення співвідношення можна запропонувати алгоритм обчислення виразу зазначеного у лівій частині рівності, що має вказану у правій частині рівності асимптотичну оцінку часу виконання програми.

2.15. Чи можна описати алгоритм для кожної з задач підрахунку суми наведених нижче, асимптотична складність яких буде $O(n)$? Якщо так, то наведіть фрагмент такої програми.

a) $\sqrt{2+\sqrt{2+\cdots+\sqrt{2}}}$, ($n$ коренів);

b) $y = x^{2^n} + x^{2^{n-1}} + \ldots + x^4 + x^2 + 1$;

c) $\left(1+\frac{1}{1^1}\right)\left(1+\frac{1}{2^2}\right)\ldots\left(1+\frac{1}{n^n}\right)$;

d) $1 + \sin x + \cdots + \sin^n x$.

2.16. Опишіть функції натурального аргументу n, час виконання яких у найгіршому випадку має асимптотику:

a) $O(n)$ b) $O(n^2)$ c) $O(n^3)$
d) $O(\log n)$ e) $O(n \log n)$ f) $O(2^n)$

2.17. Опишіть функції натуральних аргументів n та m, час виконання яких у найгіршому випадку має асимптотику:

a) $O(n+m)$ b) $O(m \log n)$ c) $O(n^m)$

2.18. Натуральне число називається паліндромом, якщо його запис читається однаково зліва направо і справа наліво (наприклад, 1, 393, 4884). Скласти програму, що визначає, чи є задане натуральне число n паліндромом асимптотична складність якої $O(n)$, де n – кількість цифр у числі.

2.19. Числами трибоначчі називається числова послідовність $\{T_k : k \geq 0\}$, задана рекурентним співвідношенням третього порядку:

$$T_0 = 0, T_1 = T_2 = 1, T_k = T_{k-1} + T_{k-2} + T_{k-3}, \quad k \geq 3.$$

Опишіть функції для обчислення T_n за допомогою рекурентного співвідношення та використовуючи рекурсію. Обчисліть асимптотичну складність кожного з варіантів. Порівняйте абсолютний час виконання (у секундах) обох варіантів для знаходження T_{10}, T_{20}, T_{50}.

2.20. Послідовністю Падована називається числова послідовність $\{P_k : k \geq 0\}$, задана рекурентним співвідношенням третього порядку:

$$P_0 = P_1 = P_2 = 1, P_k = P_{k-1} + P_{k-3}, \qquad k \geq 3.$$

Опишіть функції для обчислення P_n за допомогою рекурентного співвідношення та використовуючи рекурсію. Обчисліть асимптотичну складність кожного з варіантів. Порівняйте абсолютний час виконання (у секундах) обох варіантів для знаходження P_{10}, P_{20}, P_{50}.

ЛАБОРАТОРНА РОБОТА 3.
Лінійний та бінарний пошук.

Контрольні запитання

3.1. Які задачі допомагає розв'язати лінійний пошук? Наведіть алгоритм лінійного пошуку та його складність (у найгіршому випадку).

3.2. У чому полягає цілочисельний бінарний пошук? Для яких структур даних він може бути реалізований? Яка складність бінарного пошуку? Яка перевага бінарного пошуку у порівнянні з лінійним пошуком? Наведіть алгоритм бінарного пошуку.

Завдання для аудиторної роботи

3.1. Одиниці
https://www.e-olymp.com/uk/problems/622
На уроках інформатики вас, напевно, вчили переводити числа з одних систем числення у інші і виконувати інші подібні операції. Прийшов час продемонструвати ці знання. Знайдіть кількість одиниць у двійковому запису заданого числа.

Вхідні дані
У вхідному файлі міститься єдине ціле число n ($0 \le n \le 2000000000$).

Вихідні дані
Вихідний файл повинен містити одне число — кількість двійкових одиниць у запису числа n.

Вхідні дані #1	*Вихідні дані #1*
5	2

Вхідні дані #2	*Вихідні дані #2*
7	3

Зауваження. Скористайтеся побітовими операціями.

3.2. Реалізуйте інтерфейс для роботи з англійсько-українським словником та швидким пошуком перекладу. Реалізацію здійсніть у вигляді сукупності двох функцій описаних нижче. Функція

```
def addTranslation(eng, translation)
```

додає до словника англійське слово `eng` та його переклад `translation`. Пари (`eng, translation`) приходяться у порядку, що відповідає лексикографічному порядку. Функція

```
def find(eng)
```

повертає переклад слова eng зі словника, якщо воно міститься у словнику, або порожній рядок у іншому разі.

Тестова програма main.py, файл user.py, що містить вищезгаданий інтерфейс та вхідні дані (текстові файли) розташовані за посиланням:
https://github.com/krenevych/algo/tree/master/labs/L03/task2
Завантажте всі файли, що містяться за цим посиланням, у одну папку та реалізуйте згаданий вище інтерфейс (файл user.py). Для перевірки правильності алгоритму запустіть файл main.py.

3.3. Реалізуйте алгоритм пошуку номеру найпершого входження до заданого масиву, заданого числа x. Якщо заданий елемент відсутній у списку - поверніть номер першого елементу, що більший за число x:

```
array[i] >= x
```

Алгоритм оформіть у вигляді функції

```
def bsearch_leftmost(array, x)
```

Гарантується, що вхідний масив array впорядкований за неспаданням.

Тестова програма розташована за посиланням:
https://github.com/krenevych/algo/tree/master/labs/L03/task3
Завантажте всі файли, що містяться за цим посиланням, у одну папку. Реалізуйте згадану вище функцію, інтерфейс якої міститься у файлі user.py. Для перевірки правильності алгоритму запустіть файл main.py.

Завдання для самостійної роботи

3.4. Циклічні зсуви
https://www.e-olymp.com/uk/problems/27

3.5. Сортування за зростом
https://www.e-olymp.com/uk/problems/3607
Вказівка: Використайте лінійний пошук для підрахунку відповіді.

3.6. Затятий колекціонер метеликів
https://www.e-olymp.com/uk/problems/3966

3.7. Мутанти
https://www.e-olymp.com/uk/problems/3970

Вказівка: Використайте бінарний пошук для знаходження відповіді для кожного вказаного інтервалу.

Додаткові завдання для самостійної роботи

3.8. Реалізуйте алгоритм пошуку номеру останнього входження до заданого масиву заданого числа x. Якщо такий елемент відсутній у списку - поверніть номер останнього елементу, що менший за число x. Зауважимо, що якщо всі елементи масиву менші за шукане число, програма має повертати значення `-1` (мінус один).

`array[i] >= x`

Алгоритм оформіть у вигляді функції

```
def bsearch_rightmost(array, x)
```

Гарантується, що вхідний масив `array` впорядкований за неспаданням.

Тестова програма розташована за посиланням:
https://github.com/krenevych/algo/tree/master/labs/L03/task8
Завантажте всі файли, що містяться за цим посиланням, у одну папку. Реалізуйте згадану вище функцію, інтерфейс якої міститься у файлі `user.py`. Для перевірки правильності алгоритму запустіть файл `main.py`.

3.9. Знайдіть кількість входжень заданого числа x до масиву цілих чисел `array`. Алгоритм оформіть у вигляді функції

```
def counter(array, x)
```

Гарантується, що вхідний масив `array` впорядкований за неспаданням.

Тестова програма розташована за посиланням:
https://github.com/krenevych/algo/tree/master/labs/L03/task9
Завантажте всі файли, що містяться за цим посиланням, у одну папку. Реалізуйте згадану вище функцію, інтерфейс якої міститься у файлі `user.py`. Для перевірки правильності алгоритму запустіть файл `main.py`.

3.10. Реалізуйте алгоритм лінійного пошуку всіх елементів списку, що є степенями двійки та визначте асимптотичну складність отриманого алгоритму.

3.11. Припустимо, що список містить елементи, для яких визначена операція $\leq$ (менше або рівно) і елементи у цьому списку

розташовані у порядку зростання. Скориставшись цією властивістю списку, запропонуйте алгоритм лінійного пошуку та оцініть його час виконання. Чи буде такий алгоритм оптимальнішим ніж класичний лінійний пошук?

3.12. Реалізуйте алгоритм бінарного пошуку з використанням рекурсії.

ЛАБОРАТОРНА РОБОТА 4.
Розв’язання рівнянь методом бієкції.

Контрольні запитання

4.1. Для чого використовується дійсний бінарний пошук? Наведіть приклад задачі. Чи є відмінність у реалізації дійсного бінарного пошуку від цілочисельного? Які основні підходи застосовуються для моменту завершення пошуку? У чому їхні переваги та недоліки у порівнянні один з одним? Наведіть алгоритм дійсного бінарного пошуку.

Завдання для аудиторної роботи

4.1. Для монотонної на відрізку $[a, b]$ функції f розв'яжіть рівняння

$$f(x) = c. \quad (4.1)$$

Реалізацію здійсніть у вигляді сукупності двох функцій описаних нижче. Функція

```
def solve(f, c, a, b):
```

Для неспадної на відрізку $[a, b]$ функції f розв'язує рівняння (4.1), а функція

```
def solve_decreasing(f, c, a, b)
```

відповідно для незростаючої на відрізку $[a, b]$ функції f розв'язує рівняння (4.1). Для визначення моменту завершення пошуку використайте всі три підходи (точність по аргументу, точність по значенню та безпосереднє сусідство двох чисел з плаваючою крапкою). Порівняйте отримані результати.

Тестова програма розташована за посиланням:
https://github.com/krenevych/algo/tree/master/labs/L04/task1
Завантажте всі файли, що містяться за цим посиланням, у одну папку. Реалізуйте згадані вище функції, інтерфейси яких містяться у файлі `user.py`. Для перевірки правильності алгоритму запустіть файл `main.py`.

Завдання для самостійної роботи

4.2. Квадратний корінь
https://www.e-olymp.com/uk/problems/3968

4.3. Знайдіть найменше $x \in [0,10]$, що

$$f(x) = x^3 + x + 1 > 5$$

4.4. На відрізку $[1.6, 3]$ знайдіть корінь рівняння

$$\sin x = \frac{x}{3}.$$

4.5. На відрізку $[0,2]$ знайдіть корінь рівняння

$$x^3 + 4x^2 + x - 6.$$

ЛАБОРАТОРНА РОБОТА 5.
Бінарний пошук по відповіді.

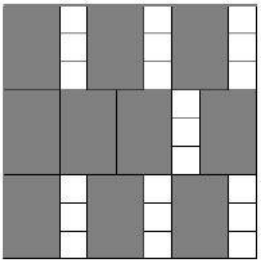
Контрольні запитання

5.1. У чому полягає концепція бінарного пошуку по відповіді?

Завдання для аудиторної роботи

5.1. Дипломи

https://www.e-olymp.com/uk/problems/3969



Коли Петя вчився в школі, він часто брав участь в олімпіадах з інформатики, математики та фізики. Так як він був досить здібним хлопчиком і старанно вчився, то на багатьох із цих олімпіад він отримував дипломи. До закінчення школи у нього накопичилося n дипломів, причому, як виявилося, всі вони мали однакові розміри: w - в ширину і h - у висоту.

Зараз Петя навчається в одному з кращих університетів і живе в гуртожитку зі своїми одногрупниками. Він вирішив прикрасити свою кімнату, повісивши на одну зі стін свої дипломи за шкільні олімпіади. Так як до бетонної стіни прикріпити дипломи досить важко, він вирішив купити спеціальну дошку з коркового дерева, щоб прикріпити її до стіни, а до неї - дипломи. Для того, щоб ця конструкція виглядала більш красиво, Петя хоче, щоб була квадратною і займала якомога менше місця на стіні. Кожен диплом повинен бути розміщений строго в прямокутнику w на h. дипломи забороняється повертати на 90 градусів. Прямокутники, які відповідають різним дипломам, не повинні мати спільних внутрішніх точок.

Потрібно написати програму, яка обчислить мінімальний розмір сторони дошки, яка буде потрібно Пете для розміщення всіх своїх дипломів.

Вхідні дані

Вхідний файл містить три цілих числа w, h, n ($1 \le w, h, n \le 10^9$).

Вихідні дані
У вихідний файл вивести відповідь до сформульованої задачі.

Вхідні дані #1	*Вихідні дані #1*
2 3 10	9

5.2. Дуже Легка Задача
https://www.e-olymp.com/uk/problems/5102
Сьогодні вранці журі вирішило додати у варіант олімпіади ще одну, Дуже Легку Задачу. Відповідальний секретар оргкомітету надрукував її умову в одному екземплярі, і тепер йому потрібно до початку олімпіади встигнути зробити ще n копій. У його розпорядженні є два ксерокси, один з яких копіює аркуш за x секунд, а другий за y. (Дозволяється використовувати як один ксерокс, так і обидва одночасно. Можна копіювати не лише з оригіналу, але і з копії).
Допоможіть йому вияснити, який мінімальний час для цього потрібно.

Вхідні дані
Три натуральних числа n, x та y ($1 \le n \le 2*10^8$, $1 \le x, y \le 10$).

Вихідні дані
Виведіть одне число – мінімальний час в секундах, необхідний для отримання n копій.

Вхідні дані #1	*Вихідні дані #1*
4 1 1	3
Вхідні дані #2	*Вихідні дані #2*
5 1 2	4

Завдання для самостійної роботи

5.3. Корови – в стійла
https://www.e-olymp.com/uk/problems/5149

5.4. Мотузочки
https://www.e-olymp.com/uk/problems/3967

5.5. Морозиво
https://www.e-olymp.com/uk/problems/4035

Завдання для самостійної роботи підвищеної складності

5.6. Черепаха
https://www.e-olymp.com/uk/problems/669

5.7. Рисосховище
https://www.e-olymp.com/uk/problems/2254

5.8. Роботи
https://www.e-olymp.com/uk/problems/161

5.9. Автобус
https://www.e-olymp.com/uk/problems/6132

ЛАБОРАТОРНА РОБОТА 6. Хешування та хеш-таблиці.

Контрольні запитання

6.1. Що таке хешування?
6.2. Наведіть приклад хеш-функції.
6.3. Наведіть основні характеристики, яким має задовольняти хеш-функція
6.4. Що таке колізія хеш-функції?
6.5. Яка хеш-функція називається ідеальною?
6.6. Наведіть приклад хеш-функції для рядків.
6.7. Що таке асоціативний масив?
6.8. Яка структура даних називається хеш-таблицею? Які основні операції вона має підтримувати?
6.9. У чому полягає процес розв'язання колізій?
6.10. Які види хеш-таблиць бувають? Які відмінності між ними?
6.11. Яким чином реалізується процес видалення ключів у хеш-таблиці з відкритою адресацією?
6.12. Поясніть концепцію методу ланцюжків для розв'язання колізій при реалізації хеш-таблиць.

Завдання для аудиторної роботи

6.1. Реалізуйте інтерфейс асоціативного масиву, ключами якого є цілі числа, а значеннями – рядки. Реалізацію здійсніть як хеш-таблицю з відкритою адресацією у вигляді сукупності функцій описаних нижче функцій. Функція

```
def init()
```

Викликається 1 раз на початку виконання програми. Функція

```
def set(key: int, value: str) -> None
```

встановлює значення `value` для ключа `key`. Якщо такий ключ відсутній у структурі – додає пару, інакше змінює значення для цього ключа. Функція

```
def get(key: int) -> str
```

за ключем `key` повертає значення зі структури (або `None`, якщо ключ відсутній у структурі). Нарешті функція

```
def delete(key: int) -> None
```

видаляє пару ключ-значення за заданим ключем, якщо такий ключ міститься у таблиці.

Тестова програма розташована за посиланням:
https://github.com/krenevych/algo/tree/master/labs/L06/task1
Завантажте всі файли, що містяться за цим посиланням, у одну папку. Реалізуйте згадані вище функції, інтерфейси яких містяться у файлі user.py. Для перевірки правильності алгоритму запустіть файл main.py.

6.2. Розв'яжіть задачу 6.1 реалізовуючи хеш-таблицю з розв'язанням колізій методом ланцюжків.

Тестова програма розташована за посиланням:
https://github.com/krenevych/algo/tree/master/labs/L06/task2
Завантажте всі файли, що містяться за цим посиланням, у одну папку. Реалізуйте згадані вище функції, інтерфейси яких містяться у файлі user.py. Для перевірки правильності алгоритму запустіть файл main.py.

Завдання для самостійної роботи

6.3. Розглядається бібліотека художніх книг. У бібліотеці може міститися кілька книг одного автора. Реалізуйте каталог для цією бібліотеки. Інтерфейс, що необхідно реалізувати наведено нижче.

```
def init():
    """ Викликається 1 раз на початку виконання програми. """
    pass

def addBook(author, title):
    """ Додає книгу до бібліотеки.
    :param author: Автор книги
    :param title: Назва книги
    """
    pass

def find(author, title):
    """ Перевіряє чи міститься задана книга у бібліотеці.
    :param author: Автор
    :param title: Назва книги
    :return: True, якщо книга міститься у бібліотеці та
             False у іншому разі. """
    return False

def delete(author, title):
    """ Видаляє книгу з бібліотеки.
    :param author: Автор
    :param title: Назва книги
```

```
    """
    pass

def findByAuthor(author):
    """ Повертає список книг заданого автора.
    Якщо бібліотека не міститься книг заданого автора,
    то підпрограма повертає порожній список.
    :param author: Автор
    :return: Список книг заданого автора у алфавітному порядку."""
    return []
```

Тестова програма розташована за посиланням:
https://github.com/krenevych/algo/tree/master/labs/L06/task3
Завантажте всі файли, що містяться за цим посиланням (у тому ж числі папку data разом з усім її вмістом), у одну папку. Реалізуйте згадані вище функції, інтерфейси яких містяться у файлі user.py. Для перевірки правильності алгоритму запустіть файл main.py.

6.4. Розв'яжіть задачу 6.3 реалізовуючи хеш-таблицю з розв'язанням колізій методом ланцюжків.
Тестова програма розташована за посиланням:
https://github.com/krenevych/algo/tree/master/labs/L06/task4
Завантажте всі файли, що містяться за цим посиланням (у тому ж числі папку data разом з усім її вмістом), у одну папку. Реалізуйте згадані вище функції, інтерфейси яких містяться у файлі user.py. Для перевірки правильності алгоритму запустіть файл main.py.

6.5. Реалізуйте для хеш-таблиці описаної в задачі 6.1 ітеративний протокол для проходження усіх ключів хеш-таблиці. Реалізацію здійсніть як хеш-таблицю з відкритою адресацією у вигляді класу, описаного нижче функцій.

```
class HashTable:

    def __init__(self):
        """ Конструктор """
        pass

    def set(self, key: int, value: str) -> None:
        """ Встановлює значення value для ключа key.
        Якщо такий ключ відсутній у структурі - додає пару,
        інакше змінює значення для цього ключа.
        :param key: Ключ
        :param value: Значення
        """
        pass
```

```
    def get(self, key: int):
        """ За ключем key повертає значення зі структури.
        :param key: Ключ
        :return: Значення, що відповідає заданому ключу або None,
        якщо ключ відсутній у структурі.
        """
        return None

    def delete(self, key: int) -> None:
        """ Видаляє пару ключ-значення за заданим ключем.
        Якщо ключ у структурі відсутній - нічого не робить.
        :param key: Ключ
        """
        pass

    def __iter__(self):
        """ Створює ітератор для проходження хеш-таблиці
        :return: Новий ітератор
        """
        return None
```

Тестова програма розташована за посиланням:
https://github.com/krenevych/algo/tree/master/labs/L06/task5
Завантажте всі файли, що містяться за цим посиланням, у одну папку. Реалізуйте усі методи згаданого вище класу, що міститься у файлі user.py. Для перевірки правильності алгоритму запустіть файл main.py.

Завдання для самостійної роботи підвищеної складності

6.6. Хеш-таблиця
https://www.e-olymp.com/uk/problems/7273

ЛАБОРАТОРНА РОБОТА 7. Хеш-таблиці.

Контрольні запитання

7.1. Що таке асоціативний масив?

7.2. Яка структура даних називається хеш-таблицею? Які основні операції вона має підтримувати?

7.3. У чому полягає процес розв'язання колізій?

7.4. Які види хеш-таблиць бувають? Які відмінності між ними?

7.5. Яким чином реалізується процес видалення ключів у хеш-таблиці з відкритою адресацією?

7.6. Запропонуйте алгоритм збільшення слотів хеш-таблиці, що використовує лінійне зондування для розв’язання колізій, для випадку, якщо фактор завантаження таблиці перевищує деякий поріг.

7.7. Поясніть концепцію методу ланцюжків для розв’язання колізій при реалізації хеш-таблиць.

Завдання для аудиторної роботи

7.1. Англо-латинський словник
https://www.e-olymp.com/uk/problems/4842

7.2. Перший словник Енді
https://www.e-olymp.com/uk/problems/1227

Завдання для самостійної роботи

7.3. Слова
https://www.e-olymp.com/uk/problems/131

Завдання для самостійної роботи підвищеної складності

7.4. Чорна скринька
https://www.e-olymp.com/uk/problems/1225

7.5. Словник нецензурних слів
https://www.e-olymp.com/uk/problems/1236

7.6. Дешифратор
https://www.e-olymp.com/uk/problems/2035

ЛАБОРАТОРНА РОБОТА 8.
Сортування – алгоритми зі складністю n^2.

Контрольні запитання

8.1. Наведіть ідею алгоритму сортування обміном (бульбашкового сортування).

8.2. Наведіть ідею алгоритму сортування вибором. У чому його перевага перед алгоритмом сортування обміном?

8.3. Наведіть ідею алгоритму сортування вставкою.

Завдання для аудиторної роботи

8.1. Реалізуйте алгоритм сортування обміном (бульбашкове сортування). Реалізацію здійсніть у вигляді функції

```
def sort(array):
    """ Сортування масиву
    :param array: Вхідний масив даних, що треба відсортувати. """
    pass  # TODO: реалізуйте відповідний алгоритм тут
```

що отримує на вхід масив `array` та сортує його.
Тестова програма розташована за посиланням:
https://github.com/krenevych/algo/tree/master/labs/L08/task1
Завантажте всі файли, що містяться за цим посиланням, у одну папку. Реалізуйте згадану вище функцію у файлі user.py. Для перевірки правильності алгоритму запустіть файл main.py.

8.2. Модифікуйте алгоритм бульбашкового сортування, реалізований під час виконання задачі 8.1, перевіркою на кожному проході чи відсортований вже масив. Порівняйте час виконання обох програм.
Тестова програма розташована за посиланням:
https://github.com/krenevych/algo/tree/master/labs/L08/task2
Завантажте всі файли, що містяться за цим посиланням, у одну папку. Реалізуйте модифікацію у файлі user.py. Для перевірки правильності алгоритму запустіть файл main.py.

8.3. Реалізуйте алгоритм сортування вибором. Реалізацію здійсніть у вигляді функції

```
def sort(array):
    """ Сортування масиву
    :param array: Вхідний масив даних, що треба відсортувати. """
    pass  # TODO: реалізуйте відповідний алгоритм тут
```

що отримує на вхід масив `array` та сортує його. Порівняйте час виконання цього алгоритму сортування з алгоритмами реалізованими у попередніх задачах.
Тестова програма розташована за посиланням:
https://github.com/krenevych/algo/tree/master/labs/L08/task3
Завантажте всі файли, що містяться за цим посиланням, у одну папку. Реалізуйте згадану вище функцію у файлі user.py. Для перевірки правильності алгоритму запустіть файл main.py.

8.4. Реалізуйте алгоритм сортування вставкою. Реалізацію здійсніть у вигляді функції

```
def sort(array):
    """ Сортування масиву
    :param array: Вхідний масив даних, що треба відсортувати. """
    pass  # TODO: реалізуйте відповідний алгоритм тут
```

що отримує на вхід масив `array` та сортує його. Порівняйте час виконання цього алгоритму сортування з алгоритмами реалізованими у попередніх задачах.
Тестова програма розташована за посиланням:
https://github.com/krenevych/algo/tree/master/labs/L08/task4
Завантажте всі файли, що містяться за цим посиланням, у одну папку. Реалізуйте згадану вище функцію у файлі user.py. Для перевірки правильності алгоритму запустіть файл main.py.

Завдання для самостійної роботи

8.5. Сортування бульбашкою
https://www.e-olymp.com/uk/problems/2663

8.6. Словник
https://www.e-olymp.com/uk/problems/5089

Завдання для самостійної роботи підвищеної складності

8.7. Бібліотечний метод
https://www.e-olymp.com/uk/problems/2664

8.8. Сортування часу
https://www.e-olymp.com/uk/problems/972

8.9. Градуйований лексикографічний порядок
https://www.e-olymp.com/uk/problems/1130

8.10. Особові справи
https://www.e-olymp.com/uk/problems/1344

8.11. Хитре сортування

https://www.e-olymp.com/uk/problems/1462

8.12. Метод мінімуму
https://www.e-olymp.com/uk/problems/2662

8.13. Бібліотека
https://www.e-olymp.com/ru/problems/4230

ЛАБОРАТОРНА РОБОТА 9.
Сортування – швидкі алгоритми.

Контрольні запитання

9.1. Наведіть ідею алгоритму сортування злиттям. Яка його асимптотична складність? Які його особливості?

9.2. Наведіть ідею алгоритму швидкого сортування. Яка його асимптотична складність? Які його особливості у порівнянні з іншими алгоритмами сортуванням?

Завдання для аудиторної роботи

9.1. Реалізуйте алгоритм сортування злиттям. Реалізацію здійсніть у вигляді функції

```
def sort(array):
    """ Сортування масиву
    :param array: Вхідний масив даних, що треба відсортувати. """
    pass  # TODO: реалізуйте відповідний алгоритм тут
```

що отримує на вхід масив `array` та сортує його. Порівняйте час виконання цього алгоритму сортування з алгоритмами реалізованими у попередній лабораторній роботі.

Тестова програма розташована за посиланням:
https://github.com/krenevych/algo/tree/master/labs/L09/task1
Завантажте всі файли, що містяться за цим посиланням, у одну папку. Реалізуйте згадану вище функцію у файлі user.py. Для перевірки правильності алгоритму запустіть файл main.py.

9.2. Реалізуйте швидкий алгоритм сортування `QuickSort`. Реалізацію здійсніть у вигляді функції

```
def sort(array):
    """ Сортування масиву
    :param array: Вхідний масив даних, що треба відсортувати. """
    pass  # TODO: реалізуйте відповідний алгоритм тут
```

що отримує на вхід масив `array` та сортує його. Порівняйте час виконання цього алгоритму сортування з алгоритмом сортування вставкою.

Тестова програма розташована за посиланням:
https://github.com/krenevych/algo/tree/master/labs/L09/task2
Завантажте всі файли, що містяться за цим посиланням, у одну

папку. Реалізуйте згадану вище функцію у файлі user.py. Для перевірки правильності алгоритму запустіть файл main.py.

9.3. Проведіть аналіз швидкодії реалізованих алгоритмів сортування для різних типів та розмірів масивів (не відсортований масив згенерований випадковим чином, масив відсортований за зростанням, масив відсортований за спаданням елементів). Для цього опишіть функції

```
def bubble_sort(array):
    """ Сортування "Бульбашкою"
    :param array: Масив (список однотипових елементів)
    """
    pass  # TODO: реалізуйте відповідний алгоритм тут

def bubble_sort_optimized(array):
    """ Модифікований алгоритм сортування "Бульбашкою"
    :param array: Масив (список однотипових елементів).
    """
    pass  # TODO: реалізуйте відповідний алгоритм тут

def selection_sort(array):
    """ Сортування вибором
    :param array: Масив (список однотипових елементів)
    :return: None
    """
    pass  # TODO: реалізуйте відповідний алгоритм тут

def insertion_sort(array):
    """ Сортування вставкою
    :param array: Масив (список однотипових елементів)
    :return: None
    """
    pass  # TODO: реалізуйте відповідний алгоритм тут

def merge_sort(array):
    """ Сортування злиттям
    :param array: Масив (список однотипових елементів)
    :return: None
    """
    pass  # TODO: реалізуйте відповідний алгоритм тут

def quick_sort(array):
    """ Швидке сортування
        :param array: Масив (список однотипових елементів)
        :return: None
        """
    pass  # TODO: реалізуйте відповідний алгоритм тут
```

кожна з яких отримує на вхід масив `array` та сортує його відповідним методом.

Тестова програма розташована за посиланням:
https://github.com/krenevych/algo/tree/master/labs/L09/task3
Завантажте всі файли, що містяться за цим посиланням, у одну папку. Реалізуйте згадані вище функції у файлі user.py. Для перевірки запустіть файл main.py.

Завдання для самостійної роботи

9.4. Сортування
https://www.e-olymp.com/uk/problems/2321

9.5. Сортування злиттям
https://www.e-olymp.com/uk/problems/4037

Завдання для самостійної роботи підвищеної складності

9.6. Ультра швидке сортування
https://www.e-olymp.com/uk/problems/1303

9.7. Станція "Сортувальна"
https://www.e-olymp.com/uk/problems/1457

ЛАБОРАТОРНА РОБОТА 10. Рекурсія, повний перебір, Метод «розділяй і володарюй».

Контрольні запитання

10.1. Які алгоритми називаються алгоритмами повного перебору?
10.2. Чи існують загальні методи побудови алгоритмів повного перебору?
10.3. Що можна сказати про асимптотичну складність алгоритмів повного перебору?
10.4. У чому полягає метод гілок та меж?
10.5. У чому полягає метод декомпозиції («розділяй і володарюй»)?

Завдання для аудиторної роботи

10.1. 23 з 5

https://www.e-olymp.com/uk/problems/1540

Основна ідея наступна. Пробігаємо по масиву вхідних чисел і беремо окремо кожне число для розгляду, яке буде поточним значенням (не можна взяти нейтральний елемент, бо для операцій +, -, * він різний). Далі пробігаємо по тим числам, що залишилися в масиві. Беремо одне з них. Також пробігаємо по можливим операціям +, -, * (в даній реалізації використовується цикл по методам __add__, __sub__, __mul__ класу int, оскільки це “дешевше” ніж використовувати функцію eval). Виконуємо відповідну операцію з поточним значенням та продовжуємо рекурсію з вже новим поточним значенням та масивом, у якого на один елемент менше. Рекурсію зупиняємо в тому випадку, коли масив залишився пустий, а поточне значення співпадає з шуканим - 23.

Будемо зчитувати з вхідного файлу та перетворювати дані в кортеж, поки кортеж не матиме всі нульові елементи. Основну задачу буде виконувати функція twenty_three, яка повертає True, якщо з даних чисел можна отримати число 23 та False інакше. Рекурсію забезпечує функція _tt, що приймає поточне значення та кортеж чисел, які залишилось опрацювати.

```
OPS = ("__add__", "__sub__", "__mul__")

def twenty_three(nums):

    def _tt(value: int, nums: tuple):
```

```
        if not nums:
            return value == 23

        for i in range(len(nums)):
            for op in OPS:
                if _tt(getattr(value, op)(nums[i]),
                       nums[:i] + nums[i+1:]) is True:
                    return True

    return any(_tt(nums[i], nums[:i] + nums[i+1:])
               for i in range(len(nums)))


if __name__ == "__main__":
    with open("input.txt") as inp:
        nums = tuple(map(int, inp.readline().split()))
        while any(nums):
            print("Possible" if twenty_three(nums)
                  else "Impossible")
            nums = tuple(map(int, inp.readline().split()))
```

10.2. Машина арифметика

https://www.e-olymp.com/uk/problems/1296

Тут знадобиться глобальна змінна max_value для зберігання максимального результату. Для вхідних даних n зручно використовувати як рядок, а m - ціле число, на яке потрібно розділити n. Виберемо 1 як початковий поточний елемент, оскільки він є нейтральним для множення. Пробігаємо по рядку n та вибираємо місце, де його потрібно відокремити на дві частини (при цьому враховуючи, що непотрібно пробігати забагато, оскільки n можна розділити лише на m частин, а якщо символів в рядку n менше за m, то будуть виконуватись зайві дії). Далі запускаємо рекурсію з поточним значенням, що є добутком старого поточного значення та першої частини рядка, другої частини рядка, так кількістю разів, на яку його залишилось розбити (ця кількість буде на 1 менше). Коли залишається єдине розбиття (m = 1) перевіряємо, чи є добуток поточного елемента та підрядка n більшим за max_value. Якщо так, встановлюємо нове значення для max_value. Крім того, згідно методу гілок там меж, можна покращити даний алгоритм. Для цього потрібно додати перевірку чи є добуток поточного елемента та рядка n меншим за max_value. Якщо це так, дану гілку перевіряти далі немає смислу.

```python
def arithmetic_masha(n: str, m: int):
    """
    :param n: натуральне число
    :param m: кількість частин, на яке розбивається число n
    :return: максимальний добуток
    """
    max_value = 0

    def _am(mult: int, sub_num: str, m: int):
        """
        :param sub_num: підрядок цифр
        :param mult: добуток
        :param m: кількість шматків, на які залишилося розбити
        """
        nonlocal max_value

        value = mult * int(sub_num)
        if value < max_value:
            return
        elif m == 1:
            max_value = value
            return

        for i in range(1, len(sub_num) - m + 2):
            _am(mult * int(sub_num[:i]), sub_num[i:], m - 1)

    _am(1, n, m)
    return max_value


if __name__ == "__main__":
    with open("input.txt") as inp:
        for line in inp:
            n, m = line.split()
            print(arithmetic_masha(n, int(m)))
```

10.3. Задача про призначення
https://www.e-olymp.com/uk/problems/1781
Будемо зберігати мінімальну можливу вартість призначення в глобальній змінній min_salary. Оскільки від порядку визначення призначення вартість не зміниться, беремо першого працівника (перший рядок) в матриці призначень matr і перебираємо його вартості призначення (елементи). Далі запускаємо рекурсію з поточною вартістю (сумою попередньої вартості та і-тої вартості

призначення matr[0][i]) та підматрицею, в якій вирізано перший рядок та і-тий стовпчик. На кожному кроці рекурсії перевіряємо: якщо поточна загальна вартість призначень більша за мінімальну, то далі занурюватись у рекурсію немає смислу і повертаємося до попереднього кроку, якщо ні, то перевіряємо, чи залишилися ще працівники та призначення, у негативному випадку, встановлюємо нову мінімальну вартість min_salary.

```
def assignment(matrix):

    min_salary = float("inf")

    def _assignment(salary: int, matr: tuple):
        nonlocal min_salary

        if salary > min_salary:
            return
        elif not matr:
            min_salary = salary
            return

        for i in range(len(matr)):
            sub_matr = tuple([r[:i] + r[i+1:] for r in matr[1:]])
            _assignment(salary + matr[0][i], sub_matr)

    _assignment(0, matrix)
    return min_salary


if __name__ == "__main__":
    with open("input.txt") as inp:
        n = int(inp.readline())
        matrix = tuple(tuple(map(int, inp.readline().split()))
                       for _ in range(n))
        print(assignment(matrix))
```

Завдання для самостійної роботи

10.4. А побалакати?
https://www.e-olymp.com/uk/problems/3606

10.5. Банк
https://www.e-olymp.com/uk/problems/7470

Завдання для самостійної роботи підвищеної складності

10.6. Печеньки
https://www.e-olymp.com/uk/problems/2633

10.7. Відпали мирних ферзів!
https://www.e-olymp.com/uk/problems/2764

10.8. Торговець
https://www.e-olymp.com/uk/problems/224

10.9. CD
https://www.e-olymp.com/uk/problems/1266

ЛАБОРАТОРНА РОБОТА 11. Рекурсія, повний перебір, Метод «розділяй і володарюй».

Контрольні запитання

11.1. У чому полягає метод гілок та меж?
11.2. У чому полягає метод декомпозиції («розділяй і володарюй»)?

Завдання для аудиторної роботи

11.1. Словник
https://www.e-olymp.com/uk/problems/364
11.2. Путівки
https://www.e-olymp.com/uk/problems/6
11.3. Міст
https://www.e-olymp.com/uk/problems/1592

Завдання для самостійної роботи

11.4. Дуже швидке множення
https://www.e-olymp.com/uk/problems/317
Вказівка: Використайте алгоритм Множення Карацуби.
11.5. Кафе "Хоботанія"
https://www.e-olymp.com/uk/problems/4746
11.6. Розподіл оцінок
https://www.e-olymp.com/uk/problems/1524

Завдання для самостійної роботи підвищеної складності

11.7. Мінне поле
https://www.e-olymp.com/uk/problems/2000
11.8. Істотні інверсії
https://www.e-olymp.com/uk/problems/7031
11.9. Шкільний бал
https://www.e-olymp.com/uk/problems/7227

ЛАБОРАТОРНА РОБОТА 12. Стек – базові алгоритми.

Контрольні запитання

12.1. Яка структура даних називається стеком?
12.2. Наведіть основні операції для роботи зі стеком.
12.3. Які основні підходи до моделювання структури даних Стек використовуються у програмуванні на практиці?
12.4. Які структури даних називаються рекурсивними?

Завдання для аудиторної роботи

12.1. Простий стек
https://www.e-olymp.com/uk/problems/6122
Вказівка: реалізацію стеку здійсніть на базі вбудованого списку або масиву

12.2. Стек необмеженого розміру
https://www.e-olymp.com/uk/problems/6124
Вказівка: реалізацію стеку здійсніть як рекурсивну структуру.

Завдання для самостійної роботи

12.3. Стек із захистом від помилок
https://www.e-olymp.com/uk/problems/6123

12.4. Мінімум на стеці
https://www.e-olymp.com/uk/problems/4259

Завдання для самостійної роботи підвищеної складності

12.5. Рейки
https://www.e-olymp.com/uk/problems/1776

ЛАБОРАТОРНА РОБОТА 13. Стек та його застосування.

Контрольні запитання

13.1. Наведіть приклади застосування стеку.
13.2. Наведіть алгоритм перевірки правильності дужкової послідовності.
13.3. Наведіть алгоритм конвертування арифметичного виразу з інфіксої у постфіксну форму зображення.
13.4. Наведіть алгоритм обчислення арифметичного виразу зображеного у інфіксній та постфіксній формах.

Завдання для аудиторної роботи

13.1. Системи числення - 1
https://www.e-olymp.com/uk/problems/5322
13.2. Баланс дужок
https://www.e-olymp.com/uk/problems/2479

Завдання для самостійної роботи

13.3. Система числення
https://www.e-olymp.com/uk/problems/7433
13.4. Дужки
https://www.e-olymp.com/uk/problems/1994
13.5. З префіксного у інфіксне
https://www.e-olymp.com/uk/problems/731
13.6. Дужки (2)
https://www.e-olymp.com/uk/problems/855

Додаткові завдання для самостійної роботи

13.7. Системи числення
https://www.e-olymp.com/uk/problems/1008
13.8. Вирази
https://www.e-olymp.com/uk/problems/3837

Завдання для самостійної роботи підвищеної складності

13.9. Система числення - перезавантаження
https://www.e-olymp.com/uk/problems/1376
13.10. Дужки
https://www.e-olymp.com/uk/problems/5205
13.11. Перетворення

https://www.e-olymp.com/uk/problems/5539

13.12. Магічні дужки
https://www.e-olymp.com/uk/problems/3013

13.13. Математики та дужки
https://www.e-olymp.com/uk/problems/4438

13.14. Дужки
https://www.e-olymp.com/uk/problems/1801

13.15. Дріб у LATEX-і
https://www.e-olymp.com/uk/problems/2039

13.16. Інопланетяни
https://www.e-olymp.com/uk/problems/2702

13.17. Яка система числення?
https://www.e-olymp.com/uk/problems/1377

13.18. Калькулятор
https://www.e-olymp.com/uk/problems/1394

13.19. Статична складність
https://www.e-olymp.com/uk/problems/1073

ЛАБОРАТОРНА РОБОТА 14. Черги та дек. Їхнє застосування.

Контрольні запитання

14.1. Яка структура даних називається чергою?
14.2. Наведіть основні операції для роботи з чергою.
14.3. Наведіть приклади реалізації черги. Яка додаткова структура даних використовується для реалізації черги як рекурсивної структури даних?
14.4. Яка структура даних називається деком (чергою з двома кінцями або двосторонньою чергою)? У чому її відмінність від черги?
14.5. Наведіть основні операції для роботи з двосторонньою чергою.
14.6. У чому полягає ідея алгоритму реалізації двосторонньої черги як рекурсивної структури? Яка додаткова структура даних при цьому використовується?
14.7. Яка структура даних називається пріоритетною чергою?

Завдання для аудиторної роботи

14.1. Проста черга
https://www.e-olymp.com/uk/problems/6125
Вказівка: реалізацію здійсніть на базі вбудованого списку або масиву

14.2. Черга необмеженого розміру
https://www.e-olymp.com/uk/problems/6127
Вказівка: реалізацію черги здійсніть як рекурсивну структуру.

14.3. Простий дек
https://www.e-olymp.com/uk/problems/6128
Вказівка: реалізацію здійсніть на базі вбудованого списку або масиву

14.4. Дек з захистом від помилок
https://www.e-olymp.com/uk/problems/6129

14.5. Задача Іосифа Флавія
https://www.e-olymp.com/uk/problems/971
Вказівка: для розв'язання задачі використайте структуру даних черга або двостороння черга реалізовану у попередніх задачах.

Завдання для самостійної роботи

14.6. Черга з захистом від помилок
https://www.e-olymp.com/uk/problems/6126

14.7. Дек необмеженого розміру
https://www.e-olymp.com/uk/problems/6130
Вказівка: реалізацію здійсніть як рекурсивну структуру.

14.8. Гра у п’яницю
https://www.e-olymp.com/uk/problems/4005

14.9. Варварські племена
https://www.e-olymp.com/uk/problems/1549

14.10. Деки на 6 мегабайтах
https://www.e-olymp.com/uk/problems/3161

Додаткові завдання для самостійної роботи

14.11. Іосиф повторюється
https://www.e-olymp.com/uk/problems/1515

Завдання для самостійної роботи підвищеної складності

14.12. Мінімум в черзі
https://www.e-olymp.com/uk/problems/694

14.13. Сортування чергами
https://www.e-olymp.com/uk/problems/2510

ЛАБОРАТОРНА РОБОТА 15. Зв’язні списки. Їхнє застосування

Контрольні запитання

15.1. Яка структура даних називається однозв’язним списком? Наведіть основні операції з цією структурою даних та ідею алгоритму її реалізації.

15.2. Яка структура даних називається двозв’язним списком? Наведіть основні операції з цією структурою даних та ідею алгоритму її реалізації.

15.3. Яка структура даних називається кільцевим списком? Наведіть основні операції з цією структурою даних та ідею алгоритму її реалізації.

15.4. Яка структура даних називається списком з поточним елементом? Наведіть основні операції з цією структурою даних та ідею алгоритму її реалізації.

Завдання для аудиторної роботи

15.1. Реалізуйте структуру даних зв'язний список з поточним елементом.

Реалізацію здійсніть у вигляді сукупності функцій описаних нижче.

```
def init()
```

Викликається один раз на початку виконання програми.

```
def empty()
```

Перевіряє чи список порожній.

```
def reset()
```

Робить перший елемент списку, поточним.

```
def next()
```

Перейти до наступного елемента – робить поточним елементом списку, елемент, що йде за поточним. Породжує виключення `StopIteration`, якщо поточний елемент є останнім у списку.

```
def current()
```

Повертає навантаження поточного елементу. Гарантується, що функція не буде викликана, якщо список порожній.

```
def insert_after(item):
```

Вставляє новий елемент у список після поточного.

Тестова програма розташована за посиланням:
https://github.com/krenevych/algo/tree/master/labs/L15/task1
Завантажте всі файли, що містяться за цим посиланням, у одну папку. Реалізуйте згадані вище функції, інтерфейси яких містяться у файлі user1.py. Для перевірки правильності алгоритму запустіть файл main.py.

15.2. Реалізуйте структуру даних зв'язний список з поточним елементом та реалізуйте додаткові операції для роботи зі списком.
Реалізацію здійсніть у вигляді сукупності функцій описаних нижче.

```
def init()
```

Викликається один раз на початку виконання програми.

```
def empty()
```

Перевіряє чи список порожній.

```
def reset()
```

Робить перший елемент списку, поточним.

```
def next()
```

Перейти до наступного елемента – робить поточним елементом списку, елемент, що йде за поточним. Породжує виключення StopIteration, якщо поточний елемент є останнім у списку.

```
def current()
```

Повертає навантаження поточного елементу. Гарантується, що функція не буде викликана, якщо список порожній.

```
def insert_after(item):
```

Вставляє новий елемент у список після поточного.

```
def insert_before(item)
```

Вставляє новий елемент у список перед поточним.

```
def delete()
```

Видаляє поточний елемент. Поточним при цьому стає наступний елемент, що йшов у списку після поточного. Якщо елемент, що видаляється був у списку останнім, то поточним стає передостанній елемент цього списку. Гарантується, що функція не буде викликана, якщо список порожній.

```
def damp():
```

Повертає масив у якому записані всі елементи поточного списку.
Тестова програма розташована за посиланням:
https://github.com/krenevych/algo/tree/master/labs/L15/task2
Завантажте всі файли, що містяться за цим посиланням, у одну папку. Реалізуйте згадані вище функції, інтерфейси яких містяться у файлі user1.py. Для перевірки правильності алгоритму запустіть файл main.py.

15.3. Реалізуйте структуру даних двобічно зв'язний список з поточним елементом. Реалізацію здійсніть у вигляді сукупності функцій описаних нижче.

```
def init()
```

Викликається один раз на початку виконання програми.

```
def empty()
```

Перевіряє чи список порожній.

```
def set_first()
```

Робить перший елемент списку поточним.

```
def set_last()
```

Робить останній елемент списку поточним.

```
def next()
```

Перейти до наступного елемента – робить поточним елементом списку, елемент, що йде за поточним. Породжує виключення `StopIteration`, якщо поточний елемент є останнім у списку.

```
def prev()
```

Перейти до попереднього елемента – робить поточним елементом елемент списку, що йде перед поточним. Породжує виключення `StopIteration`, якщо поточний елемент є першим у списку..

```
def current()
```

Повертає навантаження поточного елементу. Гарантується, що функція не буде викликана, якщо список порожній.

```
def insert_after(item):
```

Вставляє новий елемент у список після поточного.

`def insert_before(item)`
Вставляє новий елемент у список перед поточним.
`def delete()`
Видаляє поточний елемент. Поточним при цьому стає наступний елемент, що йшов у списку після поточного. Якщо елемент, що видаляється був у списку останнім, то поточним стає передостанній елемент цього списку. Гарантується, що функція не буде викликана, якщо список порожній. *Тестова програма розташована за посиланням:* *https://github.com/krenevych/algo/tree/master/labs/L15/task3* *Завантажте всі файли, що містяться за цим посиланням, у одну папку. Реалізуйте згадані вище функції, інтерфейси яких містяться у файлі user2.py. Для перевірки правильності алгоритму запустіть файл main.py.*

15.4. Реалізуйте структуру даних двобічно зв'язний список з поточним елементом та реалізуйте додаткові операції для роботи зі списком. Реалізацію здійсніть у вигляді сукупності функцій описаних нижче.

`def init()`
Викликається один раз на початку виконання програми.
`def empty()`
Перевіряє чи список порожній.
`def set_first()`
Робить перший елемент списку поточним.
`def set_last()`
Робить останній елемент списку поточним.
`def next()`
Перейти до наступного елемента – робить поточним елементом списку, елемент, що йде за поточним. Породжує виключення `StopIteration`, якщо поточний елемент є останнім у списку.
`def prev()`
Перейти до попереднього елемента – робить поточним елементом елемент списку, що йде перед поточним. Породжує

виключення StopIteration, якщо поточний елемент є першим у списку..

```
def current()
```

Повертає навантаження поточного елементу.

```
def insert_after(item)
```

Вставляє новий елемент у список після поточного.

```
def insert_before(item)
```

Вставляє новий елемент у список перед поточним.

```
def delete()
```

Видаляє поточний елемент. Поточним при цьому стає наступний елемент, що йшов у списку після поточного. Якщо елемент, що видаляється був у списку останнім, то поточним стає передостанній елемент цього списку.

```
def damp()
```

Повертає масив у якому записані всі елементи поточного списку.

```
def len()
```

Повертає кількість елементів у списку

```
def swap_prev()
```

Міняє місцями поточний елемент списку з попереднім Гарантується, що виклик функції здійснюється лише, якщо поточний елемент не перший у списку. Поточний елемент лишається не змінним

```
def swap_next()
```

Міняє місцями поточний елемент списку з наступним. Гарантується, що виклик функції здійснюється лише, якщо поточний елемент не останній у списку. Поточний елемент лишається не змінним.

Тестова програма розташована за посиланням:
https://github.com/krenevych/algo/tree/master/labs/L15/task4
Завантажте всі файли, що містяться за цим посиланням, у одну папку. Реалізуйте згадані вище функції, інтерфейси яких містяться у файлі user2.py. Для перевірки правильності алгоритму запустіть файл main.py.

Завдання для самостійної роботи

15.5. Виведення зв’язного списку
https://www.e-olymp.com/uk/problems/7452

15.6. Обертання списку
https://www.e-olymp.com/uk/problems/7453

15.7. Зміна порядку списку
https://www.e-olymp.com/uk/problems/7468

15.8. Цикл у зв’язному списку
https://www.e-olymp.com/uk/problems/7454

Завдання для самостійної роботи підвищеної складності

15.9. Вікна
https://www.e-olymp.com/uk/problems/2397

ЛАБОРАТОРНА РОБОТА 16. Дерева. Алгоритми на деревах.

Контрольні запитання

16.1. Яка структура даних називається деревом?
16.2. Що таке вершина (вузол) дерева? Які вузли називаються дітьми вузла? Батьками?
16.3. Яка вершина дерева називається коренем? Листком?
16.4. Які вузли називаються предками для заданого вузла? Які вузли називаються його нащадками?
16.5. Що таке шлях в дереві? Як визначається його довжина?
16.6. Що таке рівень вузла у дереві? Як визначається висота дерева?
16.7. Наведіть способи зображення дерев у пам'яті комп'ютера.
16.8. Поясніть ідею алгоритму пошуку в глибину (DFS) на дереві.
16.9. Наведіть ідею алгоритму пошуку в ширину (BFS) на дереві.

Завдання для аудиторної роботи

16.1. Список телефонних номерів
https://www.e-olymp.com/uk/problems/2303
Вказівка. Для розв'язання задачі реалізуйте префіксне дерево.
16.2. LCA offline (Easy)
https://www.e-olymp.com/uk/problems/2317
Вказівка: LCA - Найменший спільний предок двох вершин

Завдання для самостійної роботи

16.3. Дерево
https://www.e-olymp.com/uk/problems/2923
16.4. Корупція
https://www.e-olymp.com/uk/problems/2996
16.5. Манкунианец и цветное дерево
https://www.e-olymp.com/uk/problems/3983

Завдання для самостійної роботи підвищеної складності

16.6. Рядки у дереві
https://www.e-olymp.com/uk/problems/2925
16.7. Спільний предок – 2
https://www.e-olymp.com/uk/problems/3299
16.8. Дерево
https://www.e-olymp.com/uk/problems/3710

16.9. Дерево
https://www.e-olymp.com/uk/problems/4173
16.10. Дерево
https://www.e-olymp.com/uk/problems/4325

ЛАБОРАТОРНА РОБОТА 17. Бінарні дерева. Бінарні дерева пошуку.

Контрольні запитання

17.1. Які дерева називаються бінарними?
17.2. Як бінарні дерева зображуються у пам'яті комп'ютера?
17.3. Наведіть модифікацію алгоритмів пошуку в глибину та ширину для бінарних дерев
17.4. Наведіть означення бінарного дерева пошуку?
17.5. Які операції визначені для бінарних дерев пошуку? Яка алгоритмічна складність цих операцій?
17.6. Наведіть ідею алгоритмів вставки, пошуку і видалення вузлів для бінарних дерев пошуку.

Завдання для аудиторної роботи

17.1. Висота дерева
https://www.e-olymp.com/uk/problems/2312
17.2. Кількість елементів
https://www.e-olymp.com/uk/problems/2313
17.3. Другий максимум
https://www.e-olymp.com/uk/problems/2314
17.4. Вывод листьев
https://www.e-olymp.com/uk/problems/2316

Завдання для самостійної роботи

17.5. Дерево
https://www.e-olymp.com/uk/problems/468
17.6. Родовідне дерево
https://www.e-olymp.com/uk/problems/2242
17.7. Однакові дерева
https://www.e-olymp.com/uk/problems/7465

Додаткові завдання для самостійної роботи

17.8. Максимальна глибина Бінарного Дерева
https://www.e-olymp.com/uk/problems/7463
17.9. Мінімальна глибина Бінарного Дерева
https://www.e-olymp.com/uk/problems/7464
17.10. Симетричне дерево
https://www.e-olymp.com/uk/problems/7467

17.11. Минимум и Максимум на дереве
https://www.e-olymp.com/uk/problems/4034
17.12. Прямой обход дерева
https://www.e-olymp.com/uk/problems/4036
17.13. Обратный обход дерева
https://www.e-olymp.com/uk/problems/4038

Завдання для самостійної роботи підвищеної складності

17.14. Прямий, центрований та обернений порядок
https://www.e-olymp.com/uk/problems/1513
17.15. Створення двійкового дерева пошуку
https://www.e-olymp.com/uk/problems/1516
17.16. Створення бінарного дерева пошуку
https://www.e-olymp.com/uk/problems/3326

ЛАБОРАТОРНА РОБОТА 18. Збалансовані бінарні дерева пошуку.

Контрольні запитання

18.1. Що таке збалансоване дерево пошуку?
18.2. Яке бінарні дерева пошуку називаються ідеально збалансованими?
18.3. Які бінарні дерева називаються AVL- збалансованим?
18.4. Що таке AVL-дерево?
18.5. Що таке фактор балансу вузла у AVL-дереві?
18.6. Наведіть ідею алгоритму малого лівого обертання у AVL-дереві. Малого правого обертання?
18.7. Наведіть ідею алгоритму великого лівого обертання у AVL-дереві. Великого правого обертання?

Завдання для аудиторної роботи

18.1. Збалансоване бінарне дерево
https://www.e-olymp.com/uk/problems/7466
18.2. Двійкове дерево пошуку 1
https://www.e-olymp.com/uk/problems/4146

Завдання для самостійної роботи

18.3. Двійкове дерево пошуку 2
https://www.e-olymp.com/uk/problems/4147

Завдання для самостійної роботи підвищеної складності

18.4. Збалансоване дерево
https://www.e-olymp.com/uk/problems/2950
18.5. АВЛ-дерева
https://www.e-olymp.com/uk/problems/1846

ЛАБОРАТОРНА РОБОТА 19. Двійкова купа та пріоритетна черга.

Контрольні запитання

19.1. Які дерева називаються повними? Майже повними?
19.2. Яка структура даних називається двійковою купою?
19.3. Які операції можна здійснювати з двійковою купою? Яка алгоритмічна складність цих операцій?
19.4. На базі якої структури даних здійснюється реалізації двійкової купи?
19.5. Наведіть приклад застосування двійкової купи?

Завдання для аудиторної роботи

19.1. Хіпуй!
https://www.e-olymp.com/uk/problems/4039
19.2. Швидке сортування
https://www.e-olymp.com/uk/problems/4848
Вказівка: сортування реалізуйте за допомогою алгоритму на базі двійкової купи (heap sort).

Завдання для самостійної роботи

19.3. Куча чи ні?
https://www.e-olymp.com/uk/problems/3737
19.4. Черга з пріоритетами
https://www.e-olymp.com/uk/problems/4847

Завдання для самостійної роботи підвищеної складності

19.5. Додати все
https://www.e-olymp.com/uk/problems/1228
19.6. ДАС Черга
https://www.e-olymp.com/uk/problems/3809
19.7. Це ліва куча?
https://www.e-olymp.com/uk/problems/2919
19.8. Нумерація повного дерева
https://www.e-olymp.com/uk/problems/402
19.9. Чорний ящик
https://www.e-olymp.com/uk/problems/3358

ЛАБОРАТОРНА РОБОТА 20. Дерево відрізків.

Контрольні запитання

20.1. Яка структура даних називається деревом відрізків?
20.2. Які основні задачі розв'язують використовуючи дерево відрізків?
20.3. Наведіть ідею реалізації структури даних дерево відрізків.

Завдання для аудиторної роботи

20.1. Діма та масив
https://www.e-olymp.com/uk/problems/2941
20.2. Діма та великий масив
https://www.e-olymp.com/uk/problems/2940

Завдання для самостійної роботи

20.3. У країні невивчених уроків
https://www.e-olymp.com/uk/problems/4481
20.4. У країні невивчених уроків 2
https://www.e-olymp.com/uk/problems/4482

Завдання для самостійної роботи підвищеної складності

20.5. Дерево відрізків
https://www.e-olymp.com/uk/problems/5952
20.6. Range Variation Query
https://www.e-olymp.com/uk/problems/695
20.7. Пригоди Незнайки та його друзів
https://www.e-olymp.com/uk/problems/4496
20.8. Загадковий пристрій
https://www.e-olymp.com/uk/problems/3318
20.9. Вінні-Пух
https://www.e-olymp.com/uk/problems/4487
20.10. Вінні-Пух 2
https://www.e-olymp.com/uk/problems/4488
20.11. Троє з Простоквашино
https://www.e-olymp.com/uk/problems/4491
20.12. Троє з Простоквашино 2
https://www.e-olymp.com/uk/problems/4492
20.13. Добуток на відрізку
https://www.e-olymp.com/uk/problems/4082

20.14. Фенєчка
https://www.e-olymp.com/uk/problems/5274

20.15. Діма та масив
https://www.e-olymp.com/uk/problems/2939

ЛАБОРАТОРНА РОБОТА 21. Графи. Побудова та властивості.

Контрольні запитання

21.1. Яка структура даних називається графом.
21.2. Який зв'язок дерев з графами?
21.3. Що таке вершина (вузол) та ребро графа?
21.4. У чому різниця між порядком та розміром графа?
21.5. Який граф називається орієнтованим? Неорієнтованим?
21.6. Які вершини називаються суміжними у неорієнтованому графі? У орієнтованому?
21.7. Що таке степінь вершини у неорієнтованому графі? Що таке напівстепені входу та виходу у орієнтованому графі? Яка вершина називається витоком? Стоком?
21.8. Які основні способи зображення графів у пам'яті комп'ютера ви знаєте?

Завдання для аудиторної роботи

21.1. Від матриці суміжності до списку ребер
https://www.e-olymp.com/uk/problems/2471
21.2. Від списків суміжності до матриці суміжності
https://www.e-olymp.com/uk/problems/3982
21.3. Операції на графі
https://www.e-olymp.com/uk/problems/2472
21.4. Підрахунок кількості ребер
https://www.e-olymp.com/uk/problems/5072

Завдання для самостійної роботи

21.5. Кількість висячих вершин 1
https://www.e-olymp.com/uk/problems/5080
21.6. Півстепені вершин за списками ребер
https://www.e-olymp.com/uk/problems/5075
21.7. Степені вершин за списками ребер
https://www.e-olymp.com/uk/problems/5074
21.8. Мультиребра
https://www.e-olymp.com/uk/problems/5073
21.9. Повний граф
https://www.e-olymp.com/uk/problems/3987

Додаткові завдання для самостійної роботи

21.10. Степені вершин
https://www.e-olymp.com/uk/problems/5082

21.11. Від матриці суміжності до списку ребер – 2
https://www.e-olymp.com/uk/problems/4766

21.12. Від списку ребер до матриці суміжності – 2
https://www.e-olymp.com/uk/problems/4767

21.13. Перевірка на неорієнтовність
https://www.e-olymp.com/uk/problems/2470

21.14. Витоки та стоки
https://www.e-olymp.com/uk/problems/3986

21.15. Регулярний граф
https://www.e-olymp.com/uk/problems/5076

21.16. Напівповний граф
https://www.e-olymp.com/uk/problems/5077

Завдання для самостійної роботи підвищеної складності

21.17. Стародавній рукопис
https://www.e-olymp.com/uk/problems/610

21.18. Транзитивність орієнтовного графа
https://www.e-olymp.com/uk/problems/5079

ЛАБОРАТОРНА РОБОТА 22. Алгоритми на незважених графах. Обхід в ширину та глибину.

Контрольні запитання

22.1. Що таке маршрут у графі? Що таке шлях у графі? Який шлях називається простим? Що таке довжина шляху на графі?
22.2. Який шлях називається циклічним? Що таке цикл в графі?
22.3. Наведіть ідею алгоритму пошуку в глибину на графі.
22.4. Наведіть ідею алгоритму пошуку в ширину на графі.
22.5. Як перевірити чи містить граф цикл?

Завдання для аудиторної роботи

22.1. Обхід в ширину
https://www.e-olymp.com/uk/problems/2401
22.2. Найкоротший шлях
https://www.e-olymp.com/uk/problems/4853
22.3. Чи є цикл?
https://www.e-olymp.com/uk/problems/4004

Завдання для самостійної роботи

22.4. Маршрути в горах
https://www.e-olymp.com/uk/problems/122
22.5. Підпал
https://www.e-olymp.com/uk/problems/4369
22.6. Отримай дерево
https://www.e-olymp.com/uk/problems/978
22.7. Найкоротша відстань
https://www.e-olymp.com/uk/problems/4852

Завдання для самостійної роботи підвищеної складності

22.8. Числа
https://www.e-olymp.com/uk/problems/4007
22.9. Геть списування!
https://www.e-olymp.com/uk/problems/4002
22.10. Покриття шляхами
https://www.e-olymp.com/uk/problems/5299
22.11. Рекурсія
https://www.e-olymp.com/uk/problems/553

22.12. Ходи ферзем!
https://www.e-olymp.com/uk/problems/1098

22.13. Забавна гра
https://www.e-olymp.com/uk/problems/4050

22.14. Хрещений батько
https://www.e-olymp.com/uk/problems/5366

22.15. База даних
https://www.e-olymp.com/uk/problems/3037

ЛАБОРАТОРНА РОБОТА 23. Топологічне сортування. Зв'язність графів.

Контрольні запитання

23.1. Що таке топологічне сортування орієнтованого графа? Наведіть алгоритм.
23.2. Наведіть означення зв’язності неорієнтованого графа.
23.3. Який граф називається не зв’язним? Що таке компонента зв’язності графа?
23.4. Наведіть алгоритм пошуку компонент зв’язності неорієнтованого графа.
23.5. Чи можна узагальнити поняття зв’язності для орієнтованих графів? Які бувають види такого узагальнення?
23.6. Наведіть алгоритм перевірки орієнтованого графа на сильну зв’язність.
23.7. Наведіть алгоритм відшукання компонент сильної зв’язності орієнтованого графа.

Завдання для аудиторної роботи

23.1. Обхід у глибину
https://www.e-olymp.com/uk/problems/4000
23.2. Зв'язність
https://www.e-olymp.com/uk/problems/982
23.3. Компоненти зв`язності
https://www.e-olymp.com/uk/problems/2269

Завдання для самостійної роботи

23.4. Зв`язність графа
https://www.e-olymp.com/uk/problems/4374
23.5. Топологічне сортування
https://www.e-olymp.com/uk/problems/1948
23.6. Пошта спонсора
https://www.e-olymp.com/uk/problems/37
23.7. Ну дуже поширений метод для графів...
https://www.e-olymp.com/uk/problems/1977

Додаткові завдання для самостійної роботи

23.8. Компоненти зв`язності - 2

https://www.e-olymp.com/uk/problems/4816

Завдання для самостійної роботи підвищеної складності

23.9. Міцно зв`язаний син гір
https://www.e-olymp.com/uk/problems/1975

23.10. Місто під струмом
https://www.e-olymp.com/uk/problems/3269

23.11. Стільниковий зв`язок
https://www.e-olymp.com/uk/problems/2003

23.12. Згоряння та компоненти зв`язності
https://www.e-olymp.com/uk/problems/4371

23.13. Конденсація графу
https://www.e-olymp.com/uk/problems/1947

23.14. Конденсація графа
https://www.e-olymp.com/uk/problems/1667

23.15. Сильна зв’язність
https://www.e-olymp.com/uk/problems/2403

23.16. Водопровід
https://www.e-olymp.com/uk/problems/40

ЛАБОРАТОРНА РОБОТА 24. Застосування теорії графів до пошуку шляхів у лабіринтах.

Контрольні запитання

24.1. Наведіть основні способи зображення лабіринтів у пам'яті комп'ютера.

24.2. Чи має якісь відмінності алгоритм пошуку в ширину для лабіринтів від відповідного алгоритму для графів?

24.3. Яким чином відбувається знаходження шляху в лабіринті?

Завдання для аудиторної роботи

24.1. Площа кімнати
https://www.e-olymp.com/uk/problems/4001

Завдання для самостійної роботи

24.2. Лінії
https://www.e-olymp.com/uk/problems/1060

24.3. Видалення клітинок
https://www.e-olymp.com/uk/problems/1063

24.4. Вихід з лабіринту
https://www.e-olymp.com/uk/problems/4820

24.5. Лабіринт
https://www.e-olymp.com/uk/problems/5622

24.6. Уникайте озер
https://www.e-olymp.com/uk/problems/1058

Додаткові завдання для самостійної роботи

24.7. Lines (2)
https://www.e-olymp.com/uk/problems/1062

24.8. Грядки
https://www.e-olymp.com/uk/problems/1065

Завдання для самостійної роботи підвищеної складності

24.9. Помста Лі Чака
https://www.e-olymp.com/uk/problems/88

24.10. Підземелля
https://www.e-olymp.com/uk/problems/432

24.11. Три держави
https://www.e-olymp.com/uk/problems/1061
24.12. Шлях коня
https://www.e-olymp.com/uk/problems/1064
24.13. Переміщення коня
https://www.e-olymp.com/uk/problems/2820
24.14. Іграшковий лабіринт
https://www.e-olymp.com/uk/problems/4452
24.15. Місце зустрічі змінити неможна
https://www.e-olymp.com/uk/problems/5069
24.16. Лабіринт
https://www.e-olymp.com/uk/problems/7215
24.17. Гном і монети
https://www.e-olymp.com/uk/problems/7229
24.18. Виведи хом’ячків
https://www.e-olymp.com/uk/problems/213

ЛАБОРАТОРНА РОБОТА 25. Алгоритми на зважених графах.

Контрольні запитання

25.1. Що таке вага ребра графа? Який граф називається зваженим?

25.2. Наведіть ідею алгоритму Беллмана-Форда для відшукання найкоротшого шляху у зваженому графі.

25.3. Яким чином можна оптимізувати алгоритм Беллмана-Форда?

25.4. Наведіть ідею алгоритму Дейкстри для відшукання найкоротшого шляху у зваженому графі.

25.5. Які переваги має алгоритм Дейкстри над алгоритмом Беллмана-Форда? Які недоліки?

Завдання для аудиторної роботи

25.1. Нехай задано орієнтований зважений граф.
Знайдіть довжину найкоротшого шляху між двома заданими вузлами цього графа.
Реалізацію здійсніть у вигляді сукупності функцій описаних нижче.

```
def init(vertices, edges)
```

Викликається один раз на початку виконання програми. `vertices`, `edges` – відповідно кількість вершин та ребер у графі.

```
def addEdge(source, destination, weight)
```

Додає ребро графа, що з'єднує вершини `source` та `destination`. `weight` – вага ребра.

```
def findDistance(start, end)
```

Знаходить та повертає довжину найкоротшого шляху, між двома заданими вершинами графа. Якщо шляху між цими вершинами не існує – повертає значення -1.
Тестова програма розташована за посиланням:
https://github.com/krenevych/algo/tree/master/labs/L25/task1
Завантажте всі файли, що містяться за цим посиланням, у одну папку. Реалізуйте згадані вище функції, інтерфейси яких містяться у файлі user.py. Для перевірки правильності алгоритму запустіть файл main.py.
Вказівка: Використайте алгоритм Беллмана-Форда.

25.2. Нехай задано орієнтований зважений граф.
Знайдіть найкоротший шлях між двома заданими вузлами цього графа.
Реалізацію здійсніть у вигляді сукупності функцій описаних нижче.

```
def init(vertices, edges)
```

Викликається один раз на початку виконання програми. Параметри `vertices` та `edges` – кількість вершин та ребер у графі відповідно.

```
def addEdge(source, destination, weight)
```

Додає ребро графа, що з'єднує вершини `source` та `destination`. `weight` – вага ребра.

```
def getWay(start, end)
```

Знаходить найкоротший шлях, між двома заданими вершинами графа. Повертає список вершин шляху `start` та end або порожній список, якщо шляху між цими вершинами не існує. Якщо таких шляхів існує кілька – виведіть будь-який з них.
Тестова програма розташована за посиланням:
https://github.com/krenevych/algo/tree/master/labs/L25/task2
Завантажте всі файли, що містяться за цим посиланням, у одну папку. Реалізуйте згадані вище функції, інтерфейси яких містяться у файлі user.py. Для перевірки правильності алгоритму запустіть файл main.py.
Вказівка. Відшукання найкоротшого шляху здійсніть використовуючи алгоритм Дейкстри. Для цього скористайтеся класом PriorityQueue – пріоритетна черга, вихідний код якого розташований за посиланням:
github.com/krenevych/algo/blob/master/labs/L25/PriorityQueue.py.

Завдання для самостійної роботи

25.3. Алгоритм Дейкстри
https://www.e-olymp.com/uk/problems/1365

25.4. Найкоротший шлях
https://www.e-olymp.com/uk/problems/4856

25.5. Флойд - 1
https://www.e-olymp.com/uk/problems/974

25.6. Флойд
https://www.e-olymp.com/uk/problems/975

25.7. Флойд - існування
https://www.e-olymp.com/uk/problems/976
25.8. Червякові діри
https://www.e-olymp.com/uk/problems/1108
25.9. Заправки
https://www.e-olymp.com/uk/problems/1388
25.10. Автобуси
https://www.e-olymp.com/uk/problems/1389
25.11. Форд-Беллман
https://www.e-olymp.com/uk/problems/1453
25.12. Лабіринт знань
https://www.e-olymp.com/uk/problems/1454
25.13. Цикл
https://www.e-olymp.com/uk/problems/1455
25.14. Ліфти
https://www.e-olymp.com/uk/problems/2209
25.15. Зміна сценарію
https://www.e-olymp.com/uk/problems/7710
25.16. Слово спонсора
https://www.e-olymp.com/uk/problems/34

Додаткові завдання для самостійної роботи

25.17. Відстань між вершинами
https://www.e-olymp.com/uk/problems/625
25.18. Доставка кефірчика
https://www.e-olymp.com/uk/problems/4006
25.19. Найкоротші шляхи
https://www.e-olymp.com/uk/problems/2968
25.20. Середня відстань
https://www.e-olymp.com/uk/problems/1670

Завдання для самостійної роботи підвищеної складності

25.21. Складний тест
https://www.e-olymp.com/uk/problems/626
25.22. Мандри
https://www.e-olymp.com/uk/problems/2267
25.23. Альтернативні шляхи
https://www.e-olymp.com/uk/problems/4637

ЛАБОРАТОРНА РОБОТА 26. Алгоритм Прима побудови каркасного дерева.

Контрольні запитання

26.1. Що таке кістякове (або каркасне) дерево? Яке кістякове дерево називається мінімальним?

26.2. Наведіть ідею алгоритму Прима.

Завдання для аудиторної роботи

26.1. Мінімальний каркас
https://www.e-olymp.com/uk/problems/981

26.2. Юрський пазл
https://www.e-olymp.com/uk/problems/9876

Завдання для самостійної роботи

26.3. Дорога
https://www.e-olymp.com/uk/problems/6576

26.4. Стаціонарна телефона мережа
https://www.e-olymp.com/uk/problems/7531

26.5. День Об`єднання
https://www.e-olymp.com/uk/problems/2967

26.6. Підземні кабелі
https://www.e-olymp.com/uk/problems/1387

26.7. Арктична мережа
https://www.e-olymp.com/uk/problems/10203

26.8. Автомагістралі
https://www.e-olymp.com/uk/problems/10205

Додаткові завдання для самостійної роботи

26.9. Мінімальний каркас
https://www.e-olymp.com/uk/problems/3835

Завдання для самостійної роботи підвищеної складності

26.10. ACM змагання і порушення зв'язку
https://www.e-olymp.com/uk/problems/1107

Список літератури та додаткових джерел

1. **Креневич А.П.** Алгоритми та структури даних. Підручник. – К.: ВПЦ "Київський Університет", 2020. – 777 с.