УКРАЇНСЬКИЙ КАТОЛИЦЬКИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНИХ НАУК
Комп’ютерні науки та Системний аналіз

Назва проекту

“Власний варіант бібліотеки itertools”

Автори:
Софія Книшоїд
Ярослав Корч
Андрій Білінський
Дмитро Гребенюк
Тумоян Кірілл

20 грудня 2021

1. **Вступ.**

Нашою ціллю була реалізація власних версій функцій з бібліотеки itertools.

В процесі роботи ми покращили свої навички з побудови та оцінки алгоритмів, а також спробували використати знання з дискретної математики на практиці. Для успішного написання цього проєкту нам були особливо корисними наступні теми та поняття з дискретної математики:

— декартовий добуток, алгоритм його знаходження;

— комбінаторне розміщення, принцип пошуку розміщень

— частковий випадок перестановок

— алгоритм знаходження комбінацій без повторень

— пошук комбінацій з повтореннями

Перед початком роботи, наша команда розподілила завдання, щоб виконання проєкту було ефективнішим. Ось список підзавдань, які ми виконували:

- Тумоян Кірілл:
  - Завдання номер 2 (функція count)
  - Завдання номер 4 (функція product)
- Дмитро Гребенюк:
  - Завдання номер 5 (функція permutations)
  - Завдання номер 6 (функція combinations)
- Софія Книшоїд:
  - Завдання номер 3 (функція repeat)
  - Написання звіту, підбір та аналіз інформації
- Андрій Білінський:
  - Завдання номер 1 (функція count)
  - Оцінка оптимальності алгоритмів, часова оцінка функцій
- Ярослав Корч:

- Завдання номер 7 (функція combinations_with_replacement)
- Реалізація та оптимізація перевірок unittest; розробка перевіркового модуля

2. Псевдокод.

Загалом, було реалізовано 7 аналогів функцій бібліотеки itertools:

- count
- cycle
- repeat
- product
- permutations
- combinations
- combinations_with_replacement

1. Count (start=0, step=1)

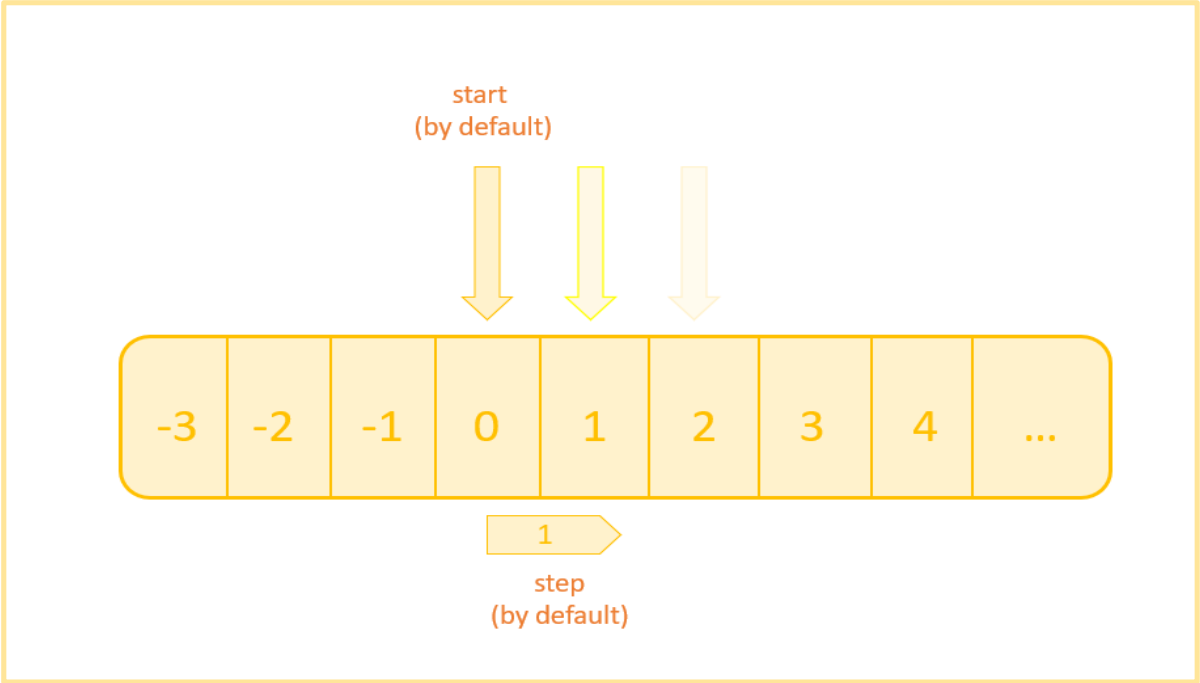
   – функція повертає нескінченний ітератор цілих чисел;

   – аргумент start визначає початок рахунку, за замовчуванням отримує значення 0;

– аргумент step визначає, наскільки відрізнятиметься від попереднього кожне наступне згенероване число, за замовчуванням отримує значення 1;

– функція може викликатися без аргументів, при цьому буде повертати послідовність виду 0, 1, 2, 3, 4, …

– функція найчастіше використовується як частина більш складних алгоритмів, передається аргументом у функції zip() та map();




```
def count(start: Union[int, float] = 0, step: Union[int, float] = 1, endpoint = None):
    """
    Returns an generator of the infinite sequence of nuberes spaced by step
    starting with start
    """
    if type(start) in (int, float) and type(step) in (int, float):
            number = start
            while True:
                yield number
                number += step
                if not endpoint is None and number > endpoint:
                    return
    raise TypeError
```

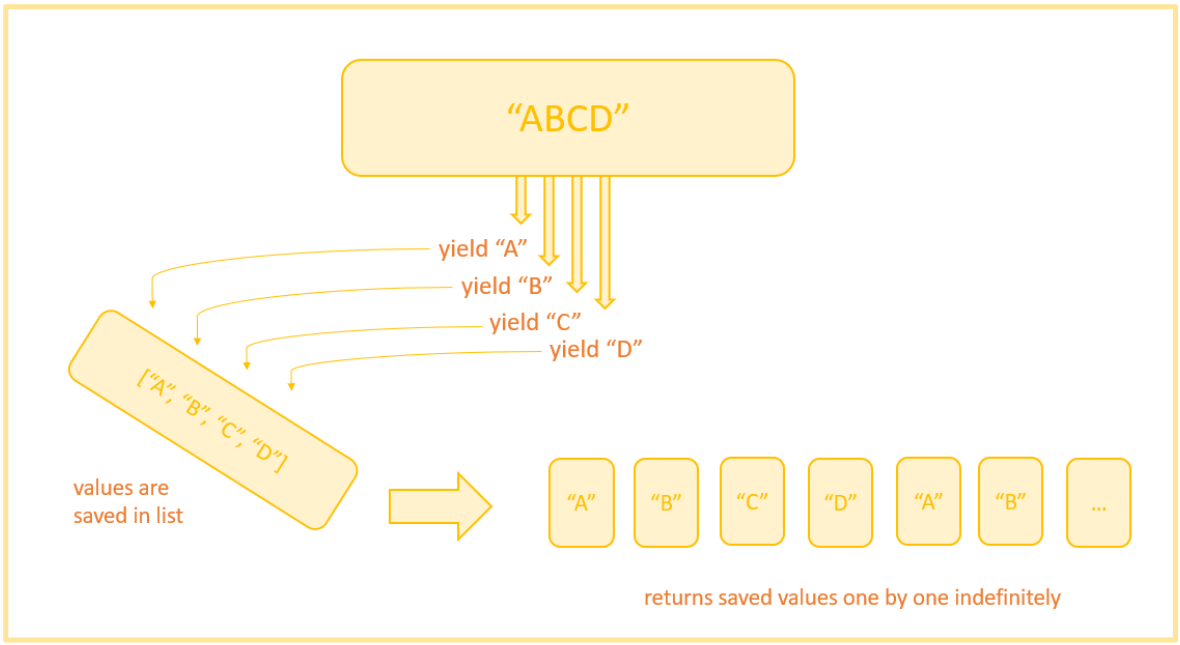
2. Цикл (iterable)

   – повертає нескінченний ітератор з елементами переданого до функції ітератора;

   – аргумент iterable представляє собою внутрішній ітератор, по якому буде проходити функція;

   – функція не може викликатися без аргументу, обов’язково потрібно передати значення iterable;

   – функція може використовувати значні об’єми пам’яті, на що слід зважати під час її використання у коді



```python
def cycle(iterable, endpoint=None):
    """
    Function for itertools.cycle()
    Makes an iterable cycle of every element.
    Returns a generator of a cycle
    :param iterable: any iterable type
    :param endpoint: optional
    for _ in cycle(iterable): print("x")
    """
    list_to_save = []
    for element in iterable:
        yield element
        if not endpoint is None:
            endpoint -= 1
            if endpoint <=0 :
                return
        list_to_save.append(element)
    while True:
        for element in list_to_save:
            yield element
            if not endpoint is None:
                endpoint -= 1
                if endpoint <=0 :
                    return
```

3. Повторення (value, stop_point=None)

   – повертає нескінченний ітератор значення value;

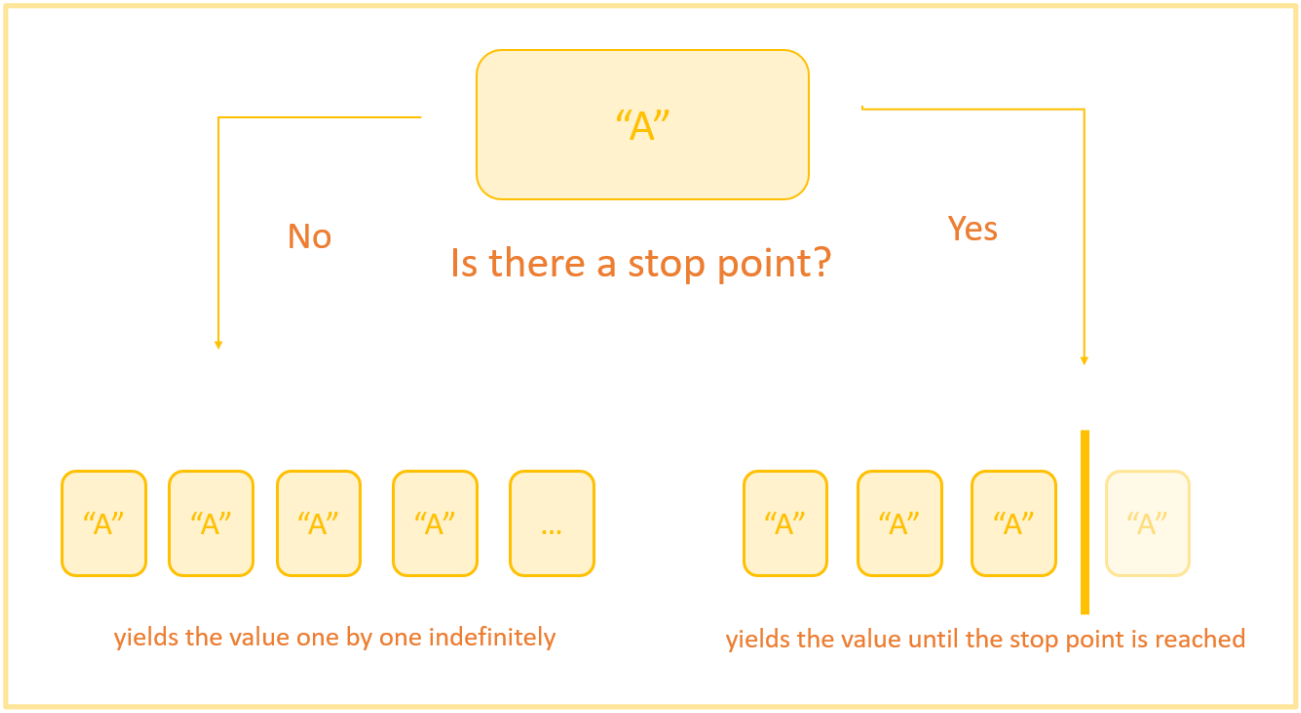
   – аргумент value представляє значення, яке щоразу повертатиме функція;

   – аргумент stop_point визначає останню ітерацію функції, з допомогою аргумента користувач регулює кількість разів повертання value;

   – аргумент value є обов’язковим для роботи функції;

   – аргумент stop_point не є обов’язковим: за замовчуванням набуває значення None, і функція повертає value нескінченну кількість разів

   – функція часто використовується як аргумент у функціях map() та zip()




```
def repeat(value, stop_point=None):
    """
    The following function mirrors the behaviour of eponymous
    itertools repeat function.
    It takes two arguments:
    --- "value" argument (a required one)
        takes a value that is to be repeated by function
    --- "stop_point" argument (an optional one)
        defines a number at which repeating should stop;
        if not passed: function repeats the value infinetely;
        by default is None
    Function either repeats the value infinitely,
    or repeats the value up until reaching the stop point.
    The function can save time and effort
    in writing long complicated tasks.
    The function is useful for "zip" and "map" usage.
    >>> for _ in repeat(8,3): print (_)
    8
    8
    8
    """
    if stop_point is not None:
        for _ in range(stop_point):
            yield value
    else:
        while True:
            yield value
```

4. Декартовий добуток (*iterables, repeat=1)

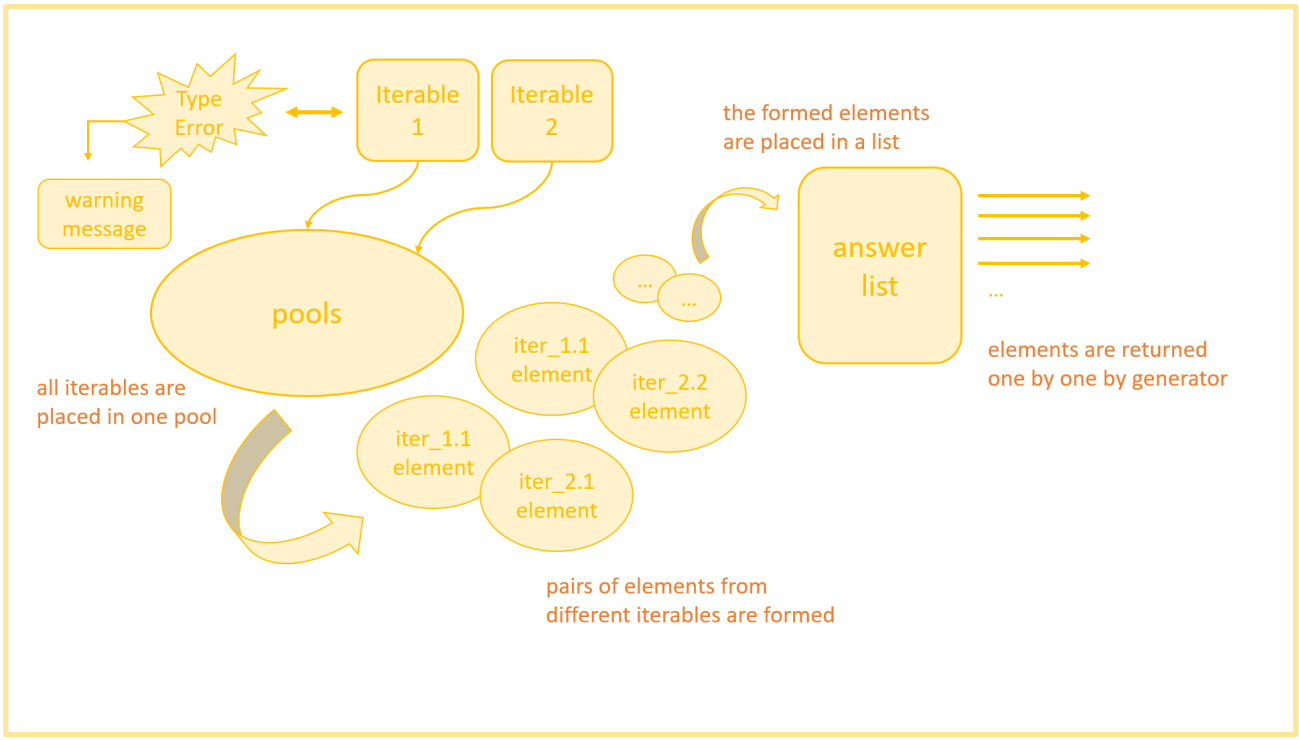
– повертає генератор декартового добутку аргументів *iterables;

– функція працює аналогічно принципу вкладених for-циклів (тобто фактично бере перший елемент із першого ітерованого об’єкта, перебирає всі з другого, міняє перший, знову перебирає…);

– аргумент repeat означає кількість повторів того самого ітерованого об’єкта; передається функції необов’язково, за замовчуванням його значення = 1;

– реалізація функції запобігає виникненню небажаної помилки TypeError та повідомляє користувача, якщо дана проблема загрожувала роботі функції;

– функція корисна не лише при використанні більш складного коду: результат її роботи дозволяє користувачу розраховувати декартовий добуток, реалізувати модель лічильника, зручно опрацьовувати об’єкти, що вимагають збереження лексикографічного порядку тощо



```python
def product(*iterables, repeat=1):
    """
    Function for itertools.product()
    Makes a generated tuple of mixed elements.
    :param iterables: optional
    :param repeat: int >= 1
    """
    # processing exceptions
    for item in iterables:
        if "__iter__" not in dir(item):
            raise TypeError
    pools = [tuple(pool) for pool in iterables] * repeat
    # making a list of tuples a repeat number of times
    lst = [[]]
    # making an empty list for answers
    for pool in pools:
        lst = [x + [y] for x in lst for y in pool]
        # appending the mixed iterable objects in the list
    for ans in lst:
        yield tuple(ans)
        # returning a generator of the ans list
```

5. Перестановки (iterable, length=None)

   – повертає генератор розміщень елементів з ітерованого об’єкта;

   – аргумент iterable представляє ітерований об’єкт, з якого будуть братися розміщення;

   – аргумент length вказує на довжину шуканих розміщень; не обов’язковий, за замовчуванням = None, тому у разі ігнорування цього аргумента шукаються перестановки елементів даного ітератора;

   – функція значною мірою спрощує роботу під час обробки великої кількості даних у задачах, що зводяться до рамок застосування комбінаторики

   – часова оцінка алгоритму:

```
original algorithm of permutations:       2.6226043701171875e-06
our version:                              2.86102294921875e-06
```

```python
def permutations(iterable, length=None):
    """
    Version of permutations without repetitions
    P(n) = n!
    iterable: (list, tuple)
    r: int
    """
    elems = tuple(iterable)
    lennn = len(iterable)
    length = lennn if not length else length
    if length > lennn:
        return
    indx = list(range(lennn))
    cycle = list(range(lennn, lennn - length, -1))
    yield tuple(elems[i] for i in indx[:length])
    while lennn:
        for ind in reversed(range(length)):
            cycle[ind] -= 1
            if cycle[ind] == 0:
                indx[ind:] = indx[ind+1:] + indx[ind:ind+1]
                cycle[ind] = lennn - ind
            else:
                j = cycle[ind]
                indx[ind], indx[-j] = indx[-j], indx[ind]
                yield tuple(elems[i] for i in indx[:length])
                break
        else:
            return
```

6. Комбінації без повторень (iterable, r)

   – повертає генератор комбінацій елементів з iterable по k;

   – аргумент r у функції вказує, по скільки елементів буде братися на комбінацію;

   – аргумент iterable є ітерованим об’єктом, з якого будуть обиратися комбінації;

   – комбінації утворені алгоритмом зберігають посортованість, тому дану функцію зручно використовувати, якщо відсортованість елементів є важливою;

   – часова оцінка алгоритму:

```
original algorithm of combinations:      2.1457672119140625e-06
our version:                             2.86102294921875e-06
```

```python
def combinations(iterable, r):
    """
    Version of combinations without repetitions
    C(n, k) = n! / ((n-k)! * k!)
    iterable: (list, tuple)
    r: int
    """
    pool = tuple(iterable)
    n = len(pool)
    if r > n:
        return
    indices = list(range(r))
    yield tuple(pool[i] for i in indices)
    while True:
        for i in reversed(range(r)):
            if indices[i] != i + n - r:
                break
        else:
            return
        indices[i] += 1
        for j in range(i+1, r):
            indices[j] = indices[j-1] + 1
        yield tuple(pool[i] for i in indices)
```

7. Комбінації з повтореннями (iterable, r)

   – функція повертає генератор комбінацій з iterable по r, до того ж елементи можуть повторюватися;

   – аргумент функції r надає алгоритму довжину підпослідовностей, комбінацій, які шукаються;

   – аргумент функції iterable надає алгоритму ітерований об’єкт, з якого будуть обиратися комбінації;

   – комбінації, отримані в результаті роботи цього алгоритму посортовані, якщо початковий аргумент iterable був також посортований

```python
def combinations_with_replacement(iterable, r: int):
    """
    This function returns r length combinations with replacrment(Order doesn't metter)
    """
    iterable = tuple(iterable)
    lenth = len(iterable)
    if r > lenth :
        raise ValueError
    indices = [0 for _ in range(r)]
    yield tuple(iterable[i] for i in indices)
    while True:
        for i in reversed(range(r)):
            if indices[i] != lenth - 1:
                break
        else:
            return
        indices[i:] = [indices[i] + 1] * (r - i)
        yield tuple(iterable[i] for i in indices)
```

8. Розміщення з повтореннями

Аналог розміщення з повтореннями у комбінаториці, написаний з використанням рекурсії.

- Повертає генератор з n^r елементами, де n – довжина масиву, r – довжина елементів.
- Елементи у кортежах неунікальні.
- Працює із ітерабельними об’єктами.

```
def permutations_repetitions(iterable, r: int):
    """
    Version of permutation with replacement.
    A(n, r) = n^r
    iterable: (list, tuple)
    r: int
    """
    if r > len(iterable): # ensures that the length of combinations is
        #less than the lenth of a iterable object
        raise ValueError

    def create_coombinations(current_coombination, r, iterable):
        for element in iterable:
            current_coombination_copy = copy.copy(current_coombination)
            current_coombination_copy.append(element)
            if len(current_coombination_copy) == r:
                yield current_coombination_copy
            else:
                yield from create_coombinations(current_coombination_copy, r, iterable)
    yield from create_coombinations([], r, iterable)
```

3. Висновки.

Якщо підсумувати результати нашої роботи над цим проєктом, можна сказати, що реалізація завдання була корисною.

Коли ми писали потрібні функції, нам дуже часто ставали у пригоді знання з дискретної математики. Можливо, використання аналогій із поняттями з цього курсу не завжди є очевидними, та вони значно спрощують роботу. Ймовірно, такі паралелі можна проводити у значно більшій кількості задач, ніж могло би здатися на перший погляд.

Виконання завдання великого обсягу показало, що тестування функцій є не менш важливою частиною роботи, ніж їх написання. Крім того, робота у групі людей зробила очевидною користь написання документації: якщо функція зрозуміло описана, значно легше розібратися у написаному коді та зрозуміти принцип дії функції.

Загалом, нам вдалося досягти поставлених цілей та реалізувати потрібні функції.