Лабораторна робота № 3

Теоретичні відомості. Комбінаторика: розміщення, перестановки, сполучення

Комбінаторика – це розділ дискретної математики, що досліджує об'єкти зі скінчених множин та операції над ними.

Основними правилами комбінаторики є *правило суми* та *правило добутку* [1, 2].

Правило суми. Якщо об'єкт *a* можна вибрати *n* способами, а інший об'єкт *b* – *m* способами, то один об'єкт *a* або *b* можна вибрати $n+m$ способами.

Правило добутку. Якщо об'єкт *a* можна вибрати *n* способами, а інший об'єкт *b* –*m* способами, то обидва об'єкти *a* і *b* у зазначеному порядку можна вибрати $n \times m$ способами.

Розглянемо основні комбінаторні об'єкти – *розміщення* та *сполучення*.

Нехай задано скінченну *n*-елементну непорожню множину $A = \{a_1, a_2, ..., a_n\}$.

Розміщенням (англ. *permutation*) з *n* елементів по *k* будемо називати *k*-елементний *впорядкований* набір елементів з множини *A*.

Сполученням (комбінацією) (англ. *combination*) з *n* елементів по *k* будемо називати *k*-елементний *невпорядкований* набір елементів з множини *A*.

Якщо в розміщенні/сполученні дозволяється наявність однакових елементів, то такі розміщення/сполучення називаються *з повтореннями*. В іншому випадку говорять про розміщення/сполучення *без повторень*.

Приклад.

Нехай задано множину $A = \{a, b, c\}$.

Тоді розміщення без повторень із трьох елементів по два ($n = 3$, $k = 2$):

$$(a,b),(a,c),(b,c),(b,a),(c,a),(c,b);$$

розміщення з повтореннями з трьох елементів по два:

$$(a,b),(a,c),(b,c),(b,a),(c,a),(c,b),(a,a),(b,b),(c,c);$$

сполучення без повторень із трьох елементів по два:

$$[a,b],[a,c],[b,c];$$

сполучення з повтореннями із трьох елементів по два:

$$[a,b],[a,c],[b,c],[a,a],[b,b],[c,c].$$

Обчислення кількості розміщень і сполучень

Кількість розміщень без повторень з *n* елементів по *k* позначатимемо як A_n^k, або $A(n,k)$, а кількість розміщень із повтореннями – $\tilde{A}_n^k$ або $\tilde{A}(n,k)$. Кількість сполучень без повторень з *n* елементів по *k* позначатимемо як C_n^k або $C(n,k)$, а кількість сполучень з повтореннями – $\tilde{C}_n^k$ або $\tilde{C}(n,k)$. У наведених позначеннях k і n – невід'ємні цілі числа, причому для розміщень та сполучень без повторення повинна виконуватись умова $k \le n$. Числа C_n^k називають ще *біноміальними коефіцієнтами* [1].

Кількість усіх розміщень та сполучень з повтореннями та без повторень, обчислюють за формулами:

$$A_n^k = n(n-1)...(n-k+1) = \frac{n!}{(n-k)!}, \tag{3.1}$$

$$\tilde{A}_n^k = n^k, \tag{3.2}$$

$$C_n^k = \frac{A_n^k}{k!} = \frac{n!}{k!(n-k)!}, \tag{3.3}$$

$$\tilde{C}_n^k = C_{n+k-1}^k = \frac{(n+k-1)!}{k!(n-1)!}. \tag{3.4}$$

Перестановка – це окремий випадок розміщень без повторень, коли з множини A вибираються всі елементи, які відрізняються тільки порядком. Згідно формули (3.1), у випадку $k = n$, кількість таких перестановок P_n становить:

$$P_n = A_n^n = n!. \tag{3.5}$$

Розглянемо тепер *перестановки з повтореннями*, тобто коли у перестановці допускаються повторення елементів. Нехай є n елементів m різних типів і елементи кожного типу однакові. Позначимо кількість елементів кожного типу $n_j \ (j = 1,\ldots,m)$. Очевидно, що $n_1 + n_2 + \ldots + n_m = n$. Тоді, кількість перестановок з повтореннями $P_n(n_1, n_2, \ldots, n_m)$ можна порахувати за формулою:

$$P_n(n_1, n_2, \ldots, n_m) = \frac{n!}{n_1! n_2! \ldots n_m!}. \tag{3.6}$$

З доведеннями формул (3.1) – (3.6) можна ознайомитися, наприклад, у [1, 2].

У формулах (3.1) – (3.6) присутня функція *факторіал*, яку можна описати наступним рекурентним співвідношенням [3]:

$$n! = \begin{cases} 1, \text{ при } n = 0 \\ n \times (n-1)!, \text{ при } n > 0 \end{cases}. \tag{3.7}$$

Обчислення факторіалу є класичним прикладом *рекурсивної функції*, тобто функції яка викликає сама себе [5, 6].

Алгоритм обчислення факторіалу (функція `factorial(n)` *– рекурсивна реалізація)*

1. Якщо аргумент функції $n = 0$, повернути число 1 як значення функції (умова зупинки рекурсії);
2. Інакше, повернути добуток числа n на значення функції від аргументу $n - 1$.

Цю ж функцію можна обчислювати ітеративним способом [5].

Хід роботи:

Частина 1. Обчислення кількості розміщень та сполучень

1. Створити нову бібліотеку `Comb` (файли `Comb.h, Comb.cpp`).
2. У бібліотеці `Comb` реалізувати функції для обчислення факторіалу, кількості розміщень без повторень та з повтореннями, кількості сполучень без повторень та з повтореннями, згідно формул (3.1) – (3.4) (`factorial(n), A(n, k), _A(n, k), C(n, k), _C(n, k)`). У заголовному файлі `Comb.h` запрограмувати інтерфейси цих функцій. Зауваження. При реалізації вищезгаданих функцій потрібно виконувати цілочисельне ділення.
3. Створити новий проект `Lab_3_1`, до якого підключити бібліотеку `Comb`. У функції `main()` проекту реалізувати обчислення кількості перестановок, розміщень та сполучень з повтореннями та без повторень за заданими значеннями параметрів *n* та *k* (значення задаються викладачем).
4. Відкомпілювати проект та запустити програму на виконання. Продемонструвати результат викладачеві.

Теоретичні відомості. Комбінаторика: генерування перестановок, сполучень та розміщень у лексикографічному порядку

Розглянемо алгоритми систематичної побудови всіх перестановок, сполучень та розміщень *n*-елементної множини $A = \{a_1, a_2, \ldots, a_n\}$ та їхнього впорядкування.

Для початку введемо поняття *лексикографічного порядку* [1, 2]. Ця назва походить від того, що в лексикографічному порядку впорядковуються слова в словнику.

Нехай маємо два кортежі: $a_1 a_2 \ldots a_n$ та $b_1 b_2 \ldots b_m$ з довжинами *n* та *m*, відповідно, і для їхніх елементів виконується одна з умов:

1. $a_1 < b_1$;

2. $a_i = b_i$ для всіх $1 \le i \le k$ і $a_{k+1} < b_{k+1}$;
3. $a_i = b_i$ для всіх $1 \le i \le n$ і $n < m$.

Тоді ці кортежі перебувають в лексикографічному порядку і кажуть, що кортеж $b_1 b_2 \ldots b_m$ є *лексикографічно наступним* після кортежу $a_1 a_2 \ldots a_n$.

Зауваження. Оскільки при генеруванні відповідні послідовності елементів мають однакову довжину, то умова 3 надалі нами розглядатися не буде.

Генерування перестановок у лексикографічному порядку

Алгоритм генерування перестановки, лексикографічно наступної після перестановки $a_1 a_2 \ldots a_n$ (функція `GenPerm(…)`*)*

1. Знайти в перестановці $a_1 a_2 \ldots a_n$ такі елементи a_i, і a_{i+1}, що $(a_i < a_{i+1}) \wedge (a_{i+1} > a_{i+2} > \ldots > a_n)$. Для цього потрібно знайти в перестановці першу справа пару сусідніх елементів, у якій елемент, що ліворуч, менший від елементу, що праворуч.
2. Переставити місцями елемент a_i та найменший з елементів серед $a_{i+1}, a_{i+2}, \ldots, a_n$, який водночас є більшим, ніж a_i.
3. Записати у висхідному порядку (посортувати за зростанням) елементи $a_{i+1}, a_{i+2}, \ldots, a_n$.

Зауваження. Функція `GenPerm(…)` повинна містити як параметр вектор (одномірний масив) що містить перестановку $a_1 a_2 \ldots a_n$, та повертати у цьому ж векторі перестановку, лексикографічно наступну після $a_1 a_2 \ldots a_n$.

Щоб побудувати всі $n!$ перестановок множини $A = \{a_1, a_2, \ldots, a_n\}$ потрібно задати першу (найменшу) перестановку $a_1 a_2 \ldots a_n$, де $a_1 < a_2 < \ldots < a_n$ і послідовно $n! - 1$ разів викликати функцію `GenPerm(…)`, для побудови лексикографічно наступних перестановок.

Зауваження. З точки зору програмування ефективніше кроки 3 і 2 алгоритму поміняти місцями. Якщо елементи $a_{i+1}, a_{i+2}, \ldots, a_n$ вже будуть відсортовані, то достатньо серед них знайти перший елемент, більший за a_i і тоді переставити їх місцями.

Також доцільно окремо розглянути випадок перестановки, де $a_{n-1} < a_n$. Наступна лексикографічна перестановка отримується просто перестановкою цих двох елементів місцями.

З обґрунтуванням описаного вище алгоритму можна познайомитися, наприклад, в [1, 2].

Кожному елементу множини *A* можна поставити у відповідність натуральне число. Таким чином, спочатку генерують відповідні перестановки елементів множини натуральних чисел $A' = \{1, 2, \ldots, n\}$, в яких потім можна замінити кожне число відповідним елементом множини *A*.

Приклад.

Розглянемо всі можливі перестановки множини $A' = \{1, 2, 3, 4, 5, 6\}$. Кількість цих перестановок $P_6 = 6! = 720$. Деякі з них наведені у таблиці 3.1.

Таблиця 3.1

Перестановки множини $\{1, 2, 3, 4, 5, 6\}$ у лексикографічному порядку

1.	1, 2, 3, 4, 5, 6	8.	1, 2, 4, 3, 6, 5
2.	1, 2, 3, 4, 6, 5	9.	1, 2, 4, 5, 3, 6
3.	1, 2, 3, 5, 4, 6	10.	1, 2, 4, 5, 6, 3
4.	1, 2, 3, 5, 6, 4	…	……
5.	1, 2, 3, 6, 4, 5	718.	6, 5, 4, 2, 3, 1
6.	1, 2, 3, 6, 5, 4	719.	6, 5, 4, 3, 1, 2
7.	1, 2, 4, 3, 5, 6	720.	6, 5, 4, 3, 2, 1

Хід роботи:

Частина 2. Генерування перестановок у лексикографічному порядку

1. У бібліотеці `Comb` реалізувати функцію `GenPerm(A,n)`, яка за даними вхідного масиву `A` довжиною `n` генерує наступну лексикографічну перестановку його елементів і повертає їх у цьому ж масиві `A`. Згідно описаного вище алгоритму (кроки 2 та 3), у функції `GenPerm(A,n)` потрібно виконати перестановку елементів місцями та сортування за зростанням елементів частини масиву:
   1.1. Варіант 1. У бібліотеку `Comb` (файл `Comb.h`) підключити створену раніше бібліотеку `Sort` (див. лабораторну роботу № 2). За необхідності переозначити тип даних `datatype` з бібліотеки `Sort` (файл `Sort.h`) відповідно до даних масиву `A`. Для перестановки елементів використати функцію `swap(…)`, а для сортування – одну з функцій `SortBubble(…)`, `SortInsertion(…)`, `SortSelection(…)` з бібліотеки `Sort` (функцію для сортування задає викладач).
   1.2. Варіант 2. У бібліотеку `Comb` (файл `Comb.h`) підключити стандартну бібліотеку `algorithm`. Після цього для сортування можна скористатися стандартною функцією `sort(…)`. Ця функція повинна приймати принаймні два параметри: елемент масиву, з якого починається сортування та елемент, перед яким сортування завершується. Наприклад, виклик `sort(A+3, A+6)` сортує елементи масиву `A` з 3-го по 5-ий. Для перестановки елементів місцями можна використати стандартну функцію `std::swap(…)`.
2. Створити новий проект `Lab_3_2`, до якого підключити бібліотеку `Comb`. У функції `main()` проекту реалізувати генерування всіх перестановок елементів заданої викладачем множини. Для цього потрібно: ініціалізувати масив `A` значеннями найменшої лексикографічної перестановки (для цілочисельного варіанту це вектор $1,2,3,\ldots,n$) та у

циклі від 2 до $n!$ викликати функції `GenPerm(A,n)` для генерації решти перестановок у лексикографічному порядку (функція для обчислення факторіалу була реалізована у першій частині даної роботи).

3. Згенерувати всі $P_n = n!$ перестановок у лексикографічному порядку та вивести результат на екран (або записати у текстовий файл). Продемонструвати результат викладачеві.

Генерування сполучень у лексикографічному порядку

Розглянемо алгоритм генерування сполучень без повторень з n елементів по k об'єктів множини $A = \{a_1, a_2, \ldots, a_n\}$ у лексикографічному порядку. Сполучення без повторень – це k-елементна підмножина множини A. Оскільки порядок запису елементів множини неістотний, то будемо записувати елементи в кожному сполученні у порядку зростання. Отже, сполучення $\{a_1, a_2, \ldots, a_k\}$ розглядатимемо як кортеж $a_1 a_2 \ldots a_k$, де $a_1 < a_2 < \ldots < a_k$.

Як і у випадку генерування перестановок, не обмежуючи загальності, розглядатимемо множину натуральних чисел $A' = \{1, 2, \ldots, n\}$.

Алгоритм генерування сполучення з n елементів по k, лексикографічно наступного після сполучення $a_1 a_2 \ldots a_k$ (функція `GenComb(…)`*)*

1. Знайти в сполучені перший справа елемент a_i такий, що $a_i \neq n - k + i$, де n – кількість елементів вихідної множини A, k – кількість елементів сполучення, i – номер (позиція) елемента a_i у кортежі $a_1 a_2 \ldots a_k$, $1 \le i \le k$. Величина $n - k + i$ дорівнює максимально можливому значенню елементу a_i в i-тій позиції.
2. Збільшити знайдений елемент на одиницю ($a_i := a_i + 1$).

3. Для кожного елементу a_j, де $i+1 \le j \le k$ встановити його значення рівне значенню попереднього елемента, збільшеного на одиницю ($a_j := a_{j-1} + 1$).

Зауваження. Як і у випадку перестановок, алгоритм будується на побудові за даним сполученням наступного, відповідно до лексикографічного порядку. Тому функція `GenComb(…)` повинна містити як параметр вектор для отримання поточного та повернення лексикографічно наступного сполучення.

Робота алгоритму повинна починатися з мінімального сполучення $\{1,2,...,k\}$

З обґрунтуванням алгоритму можна познайомитися в [1, 2].

Приклад.

Розглянемо генерування всіх сполучень з множини $A' = \{1,2,3,4,5,6\}$ з $n = 6$ по $k = 4$. Кількість цих сполучень $C_6^4 = \frac{6!}{4!2!} = 15$ і вони наведені у таблиці 3.2.

Таблиця 3.2

Сполучення з множини $\{1,2,3,4,5,6\}$ з $n = 6$ по $k = 4$

у лексикографічному порядку

1.	1,2,3,4	9.	1,3,5,6
2.	1,2,3,5	10.	1,4,5,6
3.	1,2,3,6	11.	2,3,4,5
4.	1,2,4,5	12.	2,3,4,6
5.	1,2,4,6	13.	2,3,5,6
6.	1,2,5,6	14.	2,3,5,6
7.	1,3,4,5	15.	3,4,5,6

8.	1,3,4,6		

Генерування розміщень у лексикографічному порядку

Аналогічно розглядатимемо множину натуральних чисел $A' = \{1,2,\ldots,n\}$. Використаємо алгоритм генерування лексикографічно наступного сполучення для побудови k-елементних сполучень з n-елементної множини A'. Після кожної стадії, коли побудовано чергове k-сполучення, застосуємо $k!-1$ разів алгоритм побудови перестановки за умови $n = k$ для побудови всіх перестановок елементів цього сполучення як k-елементної множини.

Хід роботи:

Частина 3. Генерування сполучень в лексиграфічному порядку

1. У бібліотеці `Comb` реалізувати, згідно описаного вище алгоритму, функцію `GenComb(A,n,k)`, яка за даними вхідного масиву `A` довжиною `k` генерує лексикографічно наступне сполучення з `n`-елементів по `k` (для множини натуральних чисел) і повертає їх у цьому ж масиві `A`.
2. У головній функції `main()` проекту `Lab_3_2` реалізувати генерування всіх сполучень з `n` по `k` множини натуральних чисел. Для цього потрібно: ініціалізувати масив `A` значеннями найменшого лексикографічного сполучення (вектор $1,2,3,\ldots,k$) та у циклі від 2 до $\frac{n!}{k!(n-k)!}$ викликати функцію `GenComb` для генерації решти сполучень у лексикографічному порядку (функція для обчислення кількості сполучень без повторень була реалізована у першій частині даної роботи).
3. Згенерувати всі $C_n^k = \frac{n!}{k!(n-k)!}$ сполучень у лексикографічному порядку та вивести результат на екран (або записати у текстовий файл). Продемонструвати результат викладачеві.