НОВ БЪЛГАРСКИ УНИВЕРСИТЕТ

Департамент Информатика

# Проект *Пермутации* към курс CSCB109 Програмиране

гл. ас. д-р Марияна Райкова mariana_sokolova@yahoo.com
гл. ас. д-р Стоян Боев stoyan@nbu.bg

## Въведение

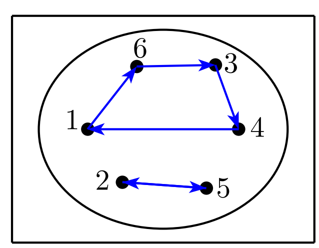
Едно от фундаменталните понятия в математиката е *функция*. В дискретната математика и в часност в компютърните науки се нуждаем от разглеждането на понятието функция на теоретико-множествено ниво. Ние използваме функции при дефинирането на дискретни структури като редици и стрингове, при оценяване времевата сложност на дадена програма и съответно ефективността на даден алгоритъм. Алгоритмите за сортиране например са пряко свързани с биекциите (взаимно-еднозначни съответствия) върху крайно множество елементи - така наречените *пермутации*. Пермутациите на $n$-елемента с дефинирана операция *композиция* ($\circ$) образуват дискретна структура, наречена *пермутационна група*, която има редица интересни свойства.

**Пример.** Нека $A = \{1, 2, 3, 4, 5, 6\}$ и функцията $f$ е дефинирана чрез таблицата:

| $x$ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $f(x)$ | 6 | 5 | 4 | 1 | 2 | 3 |

Тогава $f$ е пермутация върху множеството $A$ и записваме

$$f = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 6 & 5 & 4 & 1 & 2 & 3 \end{pmatrix}.$$




$G_f$

## Постановка

Изготвянето на проекта предполага реализация на C++ на редица функции, свързани с понятието *пермутация*.

## Пермутации

*Функция* от множеството $A$ в множеството $B$ наричаме съответствие $f$, което съпоставя на всеки елемент $a \in A$ единствен елемент $b \in B$. Елемента $b$ наричаме *образ* на $a$ и означаваме с $f(a)$, а елемента $a$ наричаме *първообраз* на $b$. Функцията означаваме с $f : A \to B$, като множеството $A$ наричаме *дефиниционна област*.

**Задача 1.** Да се дефинира двумерен динамичен масив, който има размерност 2 реда и $n$ стълба, с помощта на който ще представяме **функция** върху множество с $n$ елемента.

**Задача 2.** Да се дефинира функция на C++ за **запълване на двумерен масив** по зададено от потребителя множество от стойности на функция.

Една функция $f : A \to B$ наричаме *инективна* или *инекция*, ако тя приема различни стойности в различни точки от дефиниционната си област, т.е. ако $x, y \in A$ и $x \neq y$, то $f(x) \neq f(y)$.

**Задача 3.** Да се дефинира функция на C++, която проверява дали записана в масив функция е **инекция**.

Една функция $f : A \to B$ наричаме *сюрективна* или *сюрекция*, ако всеки елемент от множеството $B$ има първообраз.

**Задача 4.** Да се дефинира функция на C++, която проверява дали записана в масив функция е **сюрекция**.

Една функция $f : A \to B$ наричаме *биективна* или *биекция*, ако тя е едновременно инекция и сюрекция. Казваме още, че $f$ е *взаимно еднозначно съответствие* между $A$ и $B$. Ако $A$ е крайно множество, то $|A| = |B|$ и $f$ наричаме *пермутация*.

**Задача 5.** Да се дефинира функция на C++, която проверява дали записана в масив функция е **пермутация**.

Нека $n \in \mathbb{N}$. Ако с $P_n$ означим броя на пермутациите на $n$-елемента, то $P_n = n! = 1.2.3 \dots .n$.

**Задача 6.** Да се дефинира функция на C++, която намира **броя на пермутациите** на зададеното от потребителя множество.

## Неподвижни точки

Ако $f : A \to B$ и $f(x) = x$, за някое $x \in A$, то $x$ наричаме *неподвижна точка* за функцията $f$. Ако всяка точка от множеството $A$ е неподвижна, то функцията $f$ наричаме *идентитет* върху $A$ и означаваме с $\mathrm{id}_A$ или само с id, ако множеството $A$ се подразбира.

**Задача 7.** Да се дефинира функция на C++, която проверява дали записана в масив пермутация има **неподвижна точка**.

**Задача 8.** Да се дефинира функция на C++, която намира **броя на неподвижните точки** на една пермутация.

**Задача 9.** Да се дефинира функция на C++, която проверява дали записана в масив пермутация е **идентитет**.

## Независими цикли

Съществува компактен начин за представяне на една пермутация $\sigma$ върху крайно множество $S$, т. нар. *представяне като произведение на независими цикли*. Започвайки от някой произволен елемент на $S$, да речем $x$, записваме редицата

$$(x, \sigma(x), \sigma(\sigma(x)), \dots)$$

от последователни образи на $\sigma$ дотогава, докато съответния образ не е равен на $x$. Затварянето на скобата дава съответния *цикъл* на $\sigma$. След това продължаваме с избор на произволен друг елемент $y$ от $S$, който не е бил включен в някои от предходните цикли и така формираме нов цикъл включващ $y$. Тъй като $S$ е крайно множество този процес ще е краен и ни води до образуването на всички цикли на пермутацията. В случай когато дължината на един цикъл е единица (т.е. елемента е *неподвижен*) е прието за по-добра компактност да се пропуска, като се подразбира. Това представянене е еднозначно с точност до циклично пренареждане на елементите в цикъла и пренареждане на самите цикли.

**Пример.** Нека $S = \{1, 2, 3, 4, 5, 6\}$, $\sigma_1 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 3 & 1 & 5 & 4 & 6 \end{pmatrix}$, $\sigma_2 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 5 & 3 & 2 & 6 & 4 & 1 \end{pmatrix}$. Представени като произведение на независими цикли $\sigma_1$ и $\sigma_2$ имат вида:

$$\sigma_1 = (1\ 2\ 3)(4\ 5),\ \sigma_2 = (1\ 5\ 4\ 6)(2\ 3).$$

**Задача 10.** Да се дефинира функция на C++, която представя дадена пермутация във вид на **независими цикли** и ги отпечатва на екрана.

**Задача 11.** Да се дефинира функция на C++, която намира **дължините на независимите цикли**, чрез които се представя пермутацията, като тези дължини се запишат в едномерен динамичен масив.

**Задача 12.** Да се дефинира функция на C++, която получава като аргумент пермутация, представена като произведение от независими цикли и я отпечатва в табличен вид в конзолата.

## Композиция

Нека $f : A \to A$ и $g : A \to A$ са две пермутации. *Композиция* на пермутациите $f$ и $g$ означаваме с $g \circ f$ и дефинираме като пермутация от $A$ в $A$, за която е изпълнено $(g \circ f)(a) = g(f(a))$ за всяко $a \in A$.

**Пример.** Нека $\sigma_1 = (1\ 2\ 3)(4\ 5)$, $\sigma_2 = (1\ 5\ 4\ 6)(2\ 3)$. Тогава

$$\sigma_2 \circ \sigma_1 = (1\ 5\ 4\ 6)(2\ 3) \circ (1\ 2\ 3)(4\ 5) = (1\ 3\ 5\ 6)(2)(4) = (1\ 3\ 5\ 6);$$

$$\sigma_1 \circ \sigma_2 = (1\ 2\ 3)(4\ 5) \circ (1\ 5\ 4\ 6)(2\ 3) = (1\ 4\ 6\ 2)(3)(5) = (1\ 4\ 6\ 2).$$

**Задача 13.** Да се дефинира функция на C++, която намира **композицията** на две пермутации и отпечатва получената пермутация в табличен вид в конзолата.

## Обратна пермутация

За всяка пермутация $f : A \to A$ съществува единствена пермутация $g : A \to A$, за която е изпълнено:

$$g \circ f = \mathrm{id}_A \text{ и } f \circ g = \mathrm{id}_A.$$

Пермутацията $g$ наричаме *обратна* пермутация на $f$ и я означаваме с $f^{-1}$. Представянето на една пермутация като произведение от независими цикли позволява бързо пресмятане на обратните пермутации, посредством обръщане реда на всеки независим цикъл.

**Пример.** Нека $\sigma_1 = (1\ 2\ 3)(4\ 5)$, $\sigma_2 = (1\ 5\ 4\ 6)(2\ 3)$. Тогава

$$\sigma_1^{-1} = (3\ 2\ 1)(5\ 4),\ \sigma_2^{-1} = (6\ 4\ 5\ 1)(3\ 2).$$

**Задача 14.** Да се дефинира функция на C++, която намира **обратната пермутация** на дадена пермутация.

**Задача 15.** Да се дефинира функция на C++, която намира **композицията на пермутация с нейната обратна** и проверява дали полученият резултат е идентитет.

**Задача 16.** Да се дефинира функция на C++, която проверява, че ако две пермутации са различни от идентитет, т.е. $f \neq \mathrm{id}$ и $g \neq \mathrm{id}$, то операцията композиция е **некомутативна**, т.е. $f \circ g \neq g \circ f$.

## Итерация

Нека $f : A \to A$ е пермутация и $n \in \mathbb{N}_0$. Пермутацията $f^n : A \to A$ наричаме *n-та итерация* на $f$ и дефинираме като:

$$f^0 = \mathrm{id}_A, \ \ f^n = \underbrace{f \circ f \circ \cdots \circ f}_{n}.$$

От всяка пермутация можем да получим идентитет след определен брой итерации, т.е. $f^k = \mathrm{id}$, за някое $k \in \mathbb{N}_0$. Минималният брой итерации е равен на най-малкото общо кратно (НОК) на дължините на независимите цикли.

**Пример.** Нека $\sigma_1 = (1\ 2\ 3)(4\ 5)$, $\sigma_2 = (1\ 5\ 4\ 6)(2\ 3)$. Понеже $\text{НОК}(3, 2, 1) = 6$ и $\text{НОК}(4, 2) = 4$, то $\sigma_1^6 = \mathrm{id}$, $\sigma_2^4 = \mathrm{id}$.

**Задача 17.** Да се дефинира функция на C++, която намира минималното $k$, за което $f^k = \mathrm{id}$, посредством

(а) последователна проверка, дали $f^1 = \mathrm{id}$, $f^2 = \mathrm{id}$ и т.н;

(б) намиране НОК от дължините на независимите цикли на пермутацията и проверка, че след съответния брой итерации резултатът е идентитет.

В допълнение, сравнете ефктивността по отношение на време при двата подхода.

***Краен срок: 18.01.2018 г.***

***Успех!***