

# Рекурсия

Трифон Трифонов

Увод в програмирането,  
спец. Компютърни науки, 1 поток, 2018/19 г.

3–10 януари 2019 г.

# Какво е рекурсия?

- Повторение чрез позоваване на себе си
- “приятелите на моите приятели са и мои приятели”
- директориите съдържат файлове и директории
- PHP = PHP Hypertext preprocessor
- за да строшите камък:
  - ударете с чука, за да натрошите камъка на части
  - строшете получените по-малки камъни
- за да разберете какво е рекурсия, трябва да разберете какво е рекурсия

# Рекурсията в математиката

$$n! = \begin{cases} 1, & n = 0, \\ n(n-1)!, & n > 0. \end{cases}$$

$$x^n = \begin{cases} 1, & n = 0, \\ x \cdot x^{n-1}, & n > 0, \\ \frac{1}{x^{-n}}, & n < 0. \end{cases}$$

$$\gcd(a, b) = \begin{cases} a, & a = b, \\ \gcd(a - b, b), & a > b, \\ \gcd(a, b - a), & a < b. \end{cases}$$

$$f(x) = \begin{cases} 0, & x = 0, \\ f(x+1) - 1, & x > 0. \end{cases}$$

# Как се решават задачи с рекурсия?

- **Декомпозиция** — свеждане на дадена задача към множество от по-прости задачи
- Рекурсията е вид декомпозиция, при който свеждаме задача към множество от по-прости задачи **подобни на първоначалната**
- Как работи:
  - Показваме решението на най-простите задачи (**база, дъно**)
  - Показваме как по-сложна задача се свежда към една или няколко по-прости (**стъпка**)

# Математическа индукция

## Дефиниция

**Математическата индукция** е метод за доказателство, използващ като предпоставка свойството, което се доказва.

**Пример:** Да се докаже, че  $2 + 4 + \dots + 2n = n(n + 1)$ .

## Доказателство:

- за  $n = 0$ : трябва да проверим, че  $0 = 0 \cdot 1$  ✓
- нека допуснем, че сме доказали свойството за дадено  $n$
- ще го докажем за  $n + 1$ :
- $(2 + 4 + \dots + 2n) + 2(n + 1) = n(n + 1) + 2(n + 1) = (n + 1)(n + 2)$  ✓
- **Следователно:** доказахме свойството за произволно  $n$ . □

Математическата индукция е рекурсивен метод за доказателство.

# Рекурсията в програмирането

## Дефиниция

**Рекурсивна функция** наричаме функция, която извиква себе си пряко или косвено.

Рекурсивни функции се поддържат от почти всички съвременни езици за програмиране.

## Теорема

*Всяка програма с цикли може да се напише с рекурсия и обратно.*

# Примери за рекурсивни функции

Да се напише функцията, която пресмята рекурсивно:

- 1  $n!$
- 2 НОД
- 3  $x^n$
- 4 числата на Фибоначи
- 5 числата на Фибоначи, но **по-бързо**.
- 6  $\langle \text{израз} \rangle$  със скоби, където
  - $\langle \text{израз} \rangle ::= \langle \text{цифра} \rangle \mid (\langle \text{израз} \rangle \langle \text{операция} \rangle \langle \text{израз} \rangle)$
  - $\langle \text{цифра} \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$
  - $\langle \text{операция} \rangle ::= + \mid - \mid * \mid /$

# Стекови рамки на рекурсивни функции

fact	n	адрес на връщане	<code>return 1;</code>
		0	
fact	n	адрес на връщане	<code>return 1 * fact(0);</code>
		1	
fact	n	адрес на връщане	<code>return 2 * fact(1);</code>
		2	
fact	n	адрес на връщане	<code>return 3 * fact(2);</code>
		3	
fact	n	адрес на връщане	<code>return 4 * fact(3);</code>
		4	
main	n	4	<code>cout &lt;&lt; fact(4);</code>



# Примери за рекурсивни функции

Да се напише функцията, която пресмята рекурсивно:

- 1  $n!$
- 2 НОД
- 3  $x^n$
- 4 числата на Фибоначи
- 5 числата на Фибоначи, но **по-бързо**.
- 6  $\langle \text{израз} \rangle$  със скоби, където
  - $\langle \text{израз} \rangle ::= \langle \text{цифра} \rangle \mid (\langle \text{израз} \rangle \langle \text{операция} \rangle \langle \text{израз} \rangle)$
  - $\langle \text{цифра} \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$
  - $\langle \text{операция} \rangle ::= + \mid - \mid * \mid /$

# Рекурсивни функции за масиви

Да се напише функция, която чрез рекурсия

- 1 намира сума на елементите на масив
- 2 проверява дали елемент съществува в масив
- 3 проверява дали елементите на масив са подредени в растящ ред
- 4 проверява дали елементите на масив са различни
- 5 сортира масив с алгоритъма за “бързо сортиране”

# Алгоритъм за бързо сортиране

- 1 Избираме елемент от масива (“ос”)
- 2 Разделяме масива на две части:
  - елементи по-малки от оста
  - елементи по-големи или равни на оста
- 3 поставяме оста между двете части на масива
- 4 **рекурсивно** сортираме поотделно двете части на масива

Този подход за решение се нарича “разделяй и владей”.

# Задачи за търсене

Видове задачи за търсене:

- директно изброяване на кандидатите за решение
  - имаме предварително зададена последователност за обработка на всички случаи
  - **Примери:** търсене на елемент в масив, търсене на число с дадено свойство
- построяване на частични кандидати за решение
  - нямаме ясна последователност за обработка на случаите
  - **Примери:** търсене на път в лабиринт, решаване на Судоку, игра на шах

# Търсене с връщане назад (backtracking)

Търсене с **проба и грешка**:

- започваме от началната позиция
- кои са вариантите да продължим напред?
  - няколко са, правим **избор** на един от тях (**проба, стъпка напред**)
  - няма такива, отказваме се от текущия вариант и се **връщаме** да коригираме последния направен избор (**грешка, стъпка назад**)
- ако намерим търсеното решение: **успех!**
- ако изчерпим всички варианти: **провал!**

## Пример: търсене на път в лабиринт

**Задача.** Матрица от символи представя правоъгълен лабиринт:

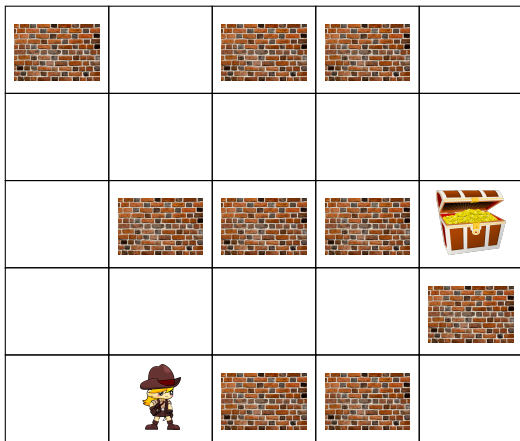
- □ — празна клетка
- \* — стена
- \$ — съкровище

Можем ли да стигнем до съкровището?

**Решение:**

- започваме от началната позиция
- оглеждаме се на север, изток, юг и запад
  - **избираме** една от посоките и стъпваме там, ако можем (**проба**)
  - като изчерпим всички посоки се **връщаме** на предишното кръстовище да изберем нова посока (**грешка**)
- ако стигнем до съкровището: **успех!**
- ако се върнем обратно в началото: **провал!**

## Търсене на съкровище в лабиринт



# Рекурсията: предимства и недостатъци

## Предимства:

- ✓ силна изразителност
- ✓ хубави математически свойства
- ✓ удобна за задачи с рекурсивна постановка
- ✓ удобна за търсене с връщане назад (backtracking)
- ✓ удобна за алгоритми от тип “разделяй и владей”

## Недостатъци:

- × изглежда объркваща и сложна за неопитни програмисти
- × скрито използване на памет за стекови рамки
- × може да е неефективна при неправилно използване
- × понякога има нужда да пишем помощни функции