

# Рекурсия. Перебор. Методы сокращения перебора

Автор: Заливако Сергей  
Сергеевич  
выпускник БГУИР

# Рекурсия. Определение

- В программировании рекурсия — вызов функции (процедуры) из неё же самой, непосредственно (простая рекурсия) или через другие функции (сложная или косвенная рекурсия), например, функция А вызывает функцию В, а функция В — функцию А.
- Количество вложенных вызовов функции или процедуры называется глубиной рекурсии.
- Преимущество рекурсивного определения объекта заключается в том, что такое конечное определение теоретически способно описывать бесконечно большое число объектов. С помощью рекурсивной программы же возможно описать бесконечное вычисление, причём без явных повторений частей программы.

# Рекурсия. Основные положения

- Рекурсию надо использовать там, где она реально необходима.
- Числа Фибоначчи и факториалы – плохой пример использования рекурсии
- Рекурсия – это всего лишь вызов подпрограммы в самой себе
- Рекурсия используется для разбиения задачи на подзадачи и решения задачи с объемом меньше, чем исходная.

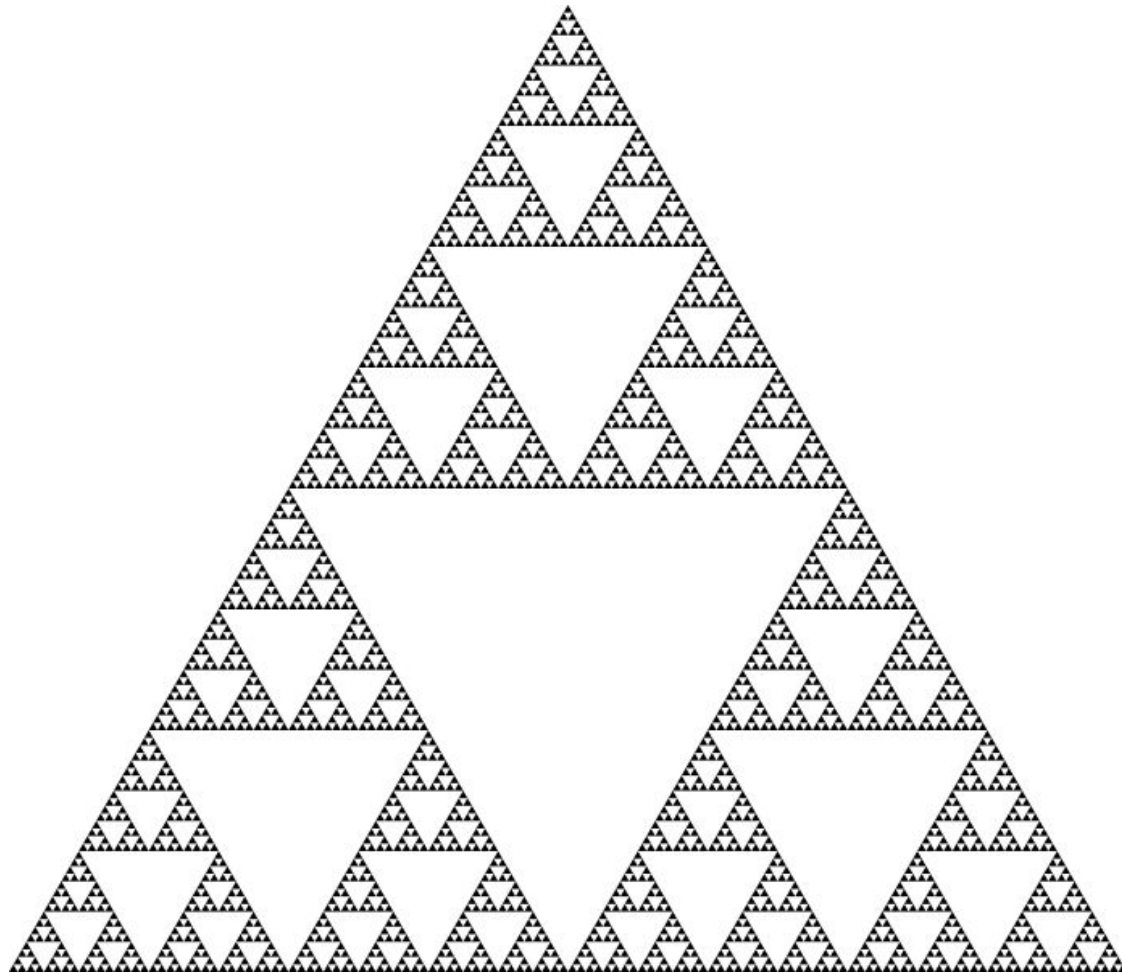
# Примеры использования рекурсии

- Поиск в глубину в графе
- Сортировка слиянием
- «Быстрая» сортировка (Хоара)
- Обход различного рода деревьев (в повседневной жизни – дерево каталогов)
- Практически незаменима в переборных задачах

# Стек вызовов

- Рекурсия использует системный стек для запоминания вызываемых подпрограмм и их параметров
- Следите за стеком.
- Изменение размера стека:
  - Pascal: {\$M <размер стека в байтах>, <максимальный размер стека>}
  - C++: #pragma comment(linker, "/STACK:<размер стека в байтах>")

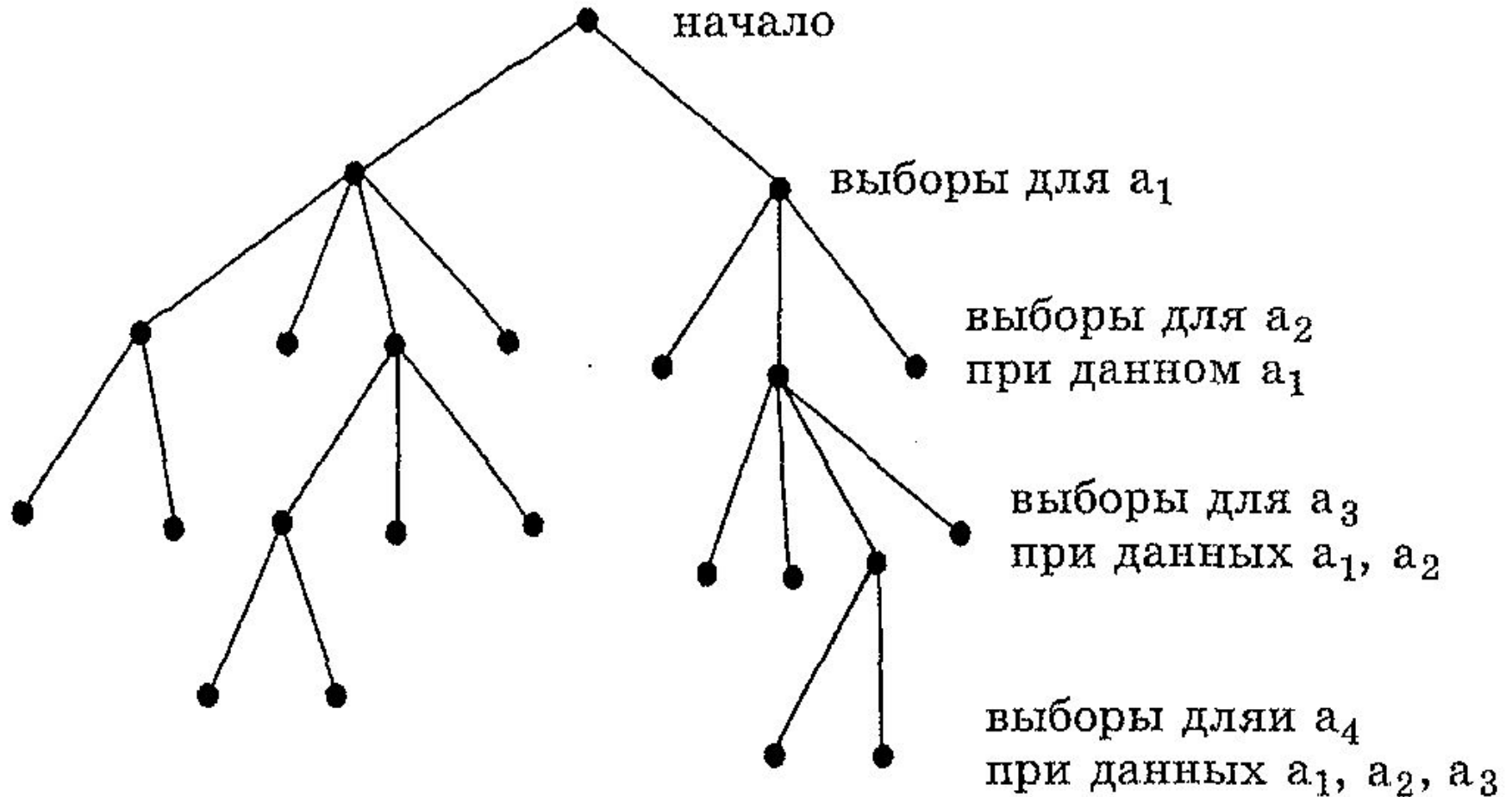
# Рекурсия. Иллюстрация



# Перебор с помощью рекурсии

- Даны  $N$  упорядоченных множеств  $U_1, U_2, \dots, U_N$  ( $N$  – неизвестно)
- Требуется построить вектор  $A = (a_1, a_2, \dots, a_N)$
- $A_1 \in U_1, A_2 \in U_2, \dots, A_N \in U_N$
- В алгоритме перебора вектор строится покомпонентно слева направо

# Перебор с помощью рекурсии. Общая схема





# Перебор с помощью рекурсии. Схема реализации

```
Procedure Backtrack (<вектор, i>);  
Begin  
  If <вектор является решением>  
    Then <записать его>  
  Else Begin  
    <вычислить  $S_i$ >;  
    For <a  $\in S_i$ >Do  
      Backtrack (<вектор + a>, i+1);  
    End;  
End;
```

# Задача о Ханойских башнях.

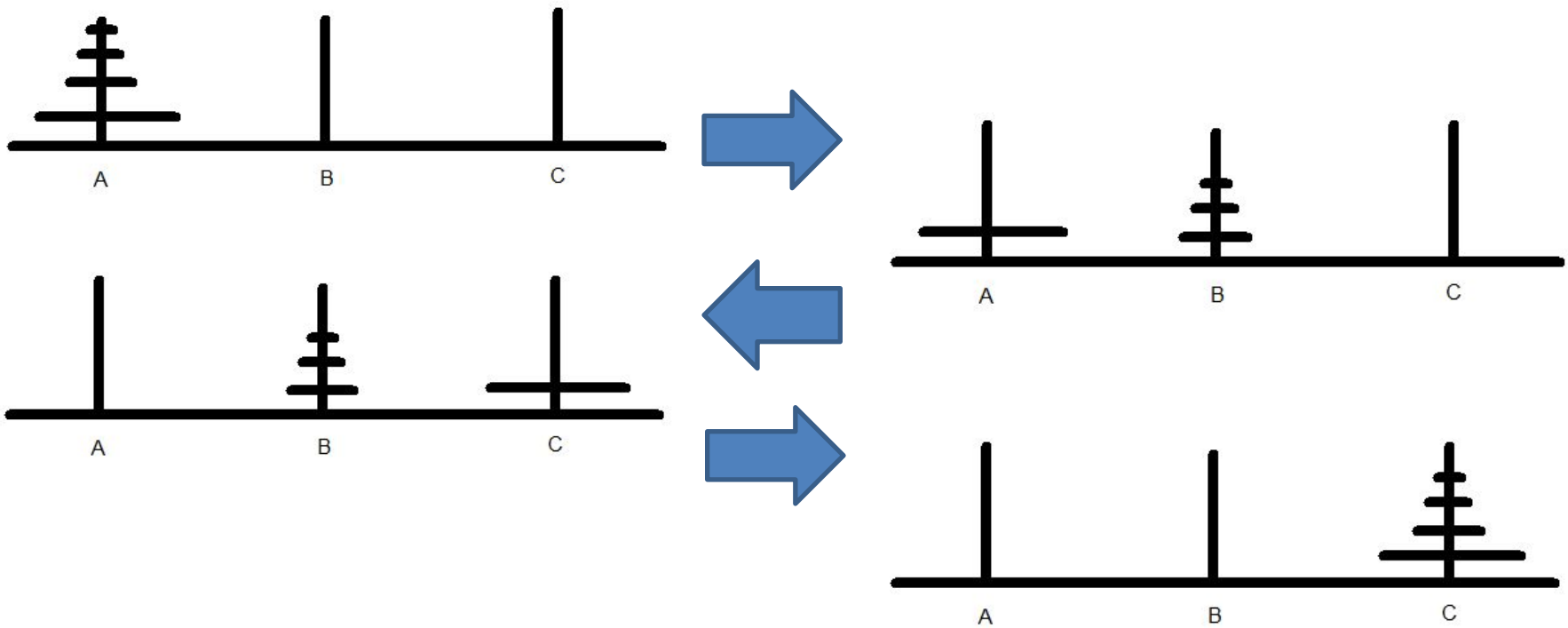
## История

- Древняя индийская легенда
- 1883 г. Франсуа Люка «Профессор Клаус»
- Современное название головоломки

# Ханойские башни. Решение

- Допустим на штыре  $n$  дисков
- Необходимо каким-то образом(пока непонятно каким) перенести  $n-1$  дисков на промежуточный штырь
- Перенесем  $n$ -й диск на последний штырь
- Таким же образом как и во втором шаге перенести  $n-1$  дисков на последний штырь

# Ханойские башни. Графическая иллюстрация решения



# Ханойские башни. Алгоритм решения.

Функция Перенести\_диск(номер\_1, номер\_2, количество)

begin

если (количество > 0) то begin

номер\_промежуточный = 6 - номер\_1 - номер\_2;

Перенести\_диск(номер\_1, номер\_промежуточный,  
количество - 1);

Вывести\_действие(номер\_1, номер\_2);

Перенести\_диск(номер\_промежуточный, номер\_1,  
количество - 1);

end;

end;

# Меморизация. Предпосылки

- При реализации рекурсивных подпрограмм часто вызываются подпрограммы с одними и теми же параметрами, т.е. выполняется «лишняя» работа
- Такая особенность рекурсии уменьшает эффективность

# Меморизация. Что это?

- От английского слова memo – памятка.
- Идея заключается в том, чтобы запомнить параметры уже вызывавшихся подпрограмм
- В случае если такие параметры повторятся, то не вызывать подпрограмму

# Меморизация. Особенности

- Эффективна, когда рекурсивная процедура или функция имеет целые параметры с небольшим диапазоном значений
- Тогда для их хранения достаточно  $n$ -мерного ( $n$  – число параметров функции) булевского массива
- Если параметры имеют сложный вид, то необходимы сложные структуры данных, что вряд ли оправданно



**Спасибо за внимание!**

Вопросы?