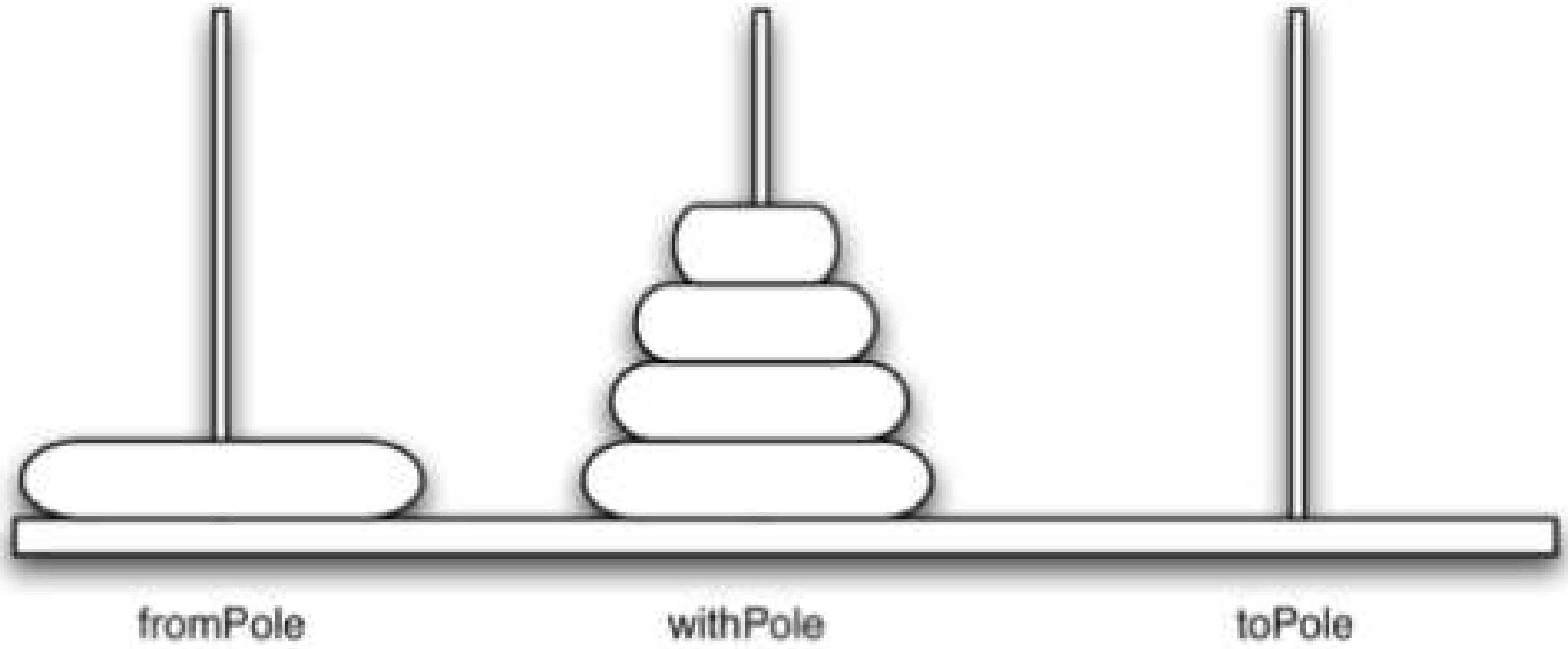
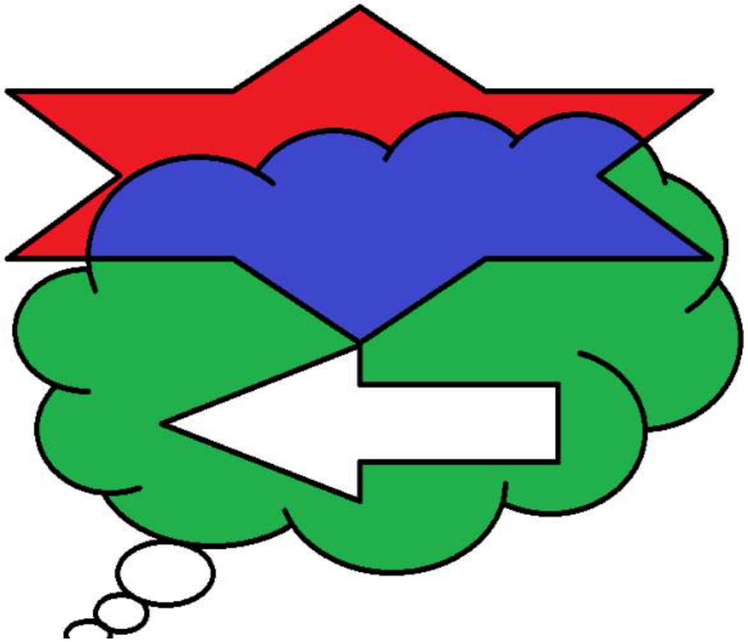
Tower of Hanoi

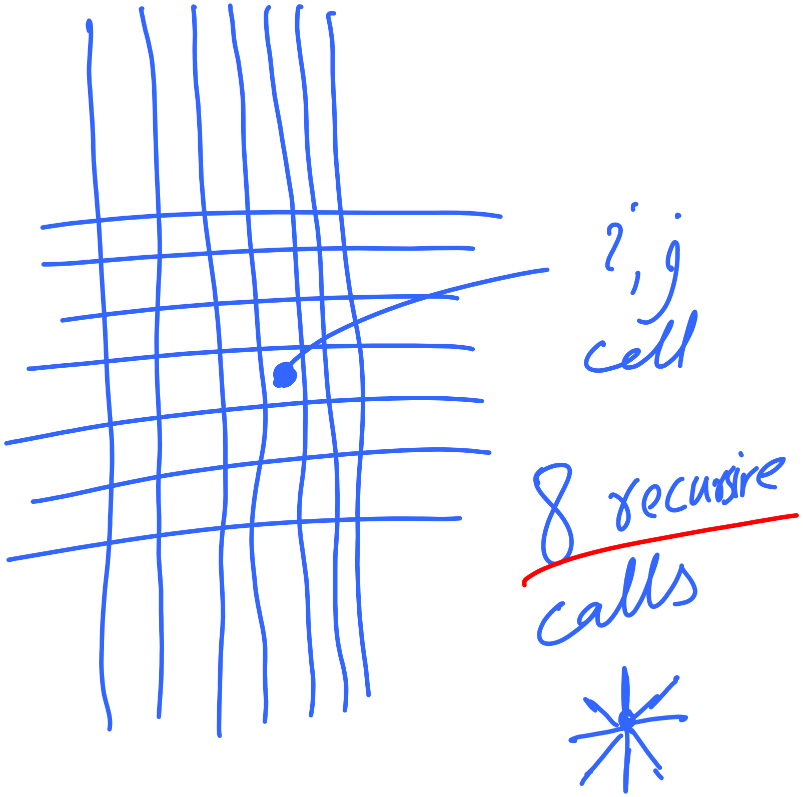
The Tower of Hanoi puzzle was invented by the French mathematician Edouard Lucas in 1883. He was inspired by a legend that tells of a Hindu temple where the puzzle was presented to young priests. At the beginning of time, the priests were given three poles and a stack of 64 gold disks, each disk a little smaller than the one beneath it. Their assignment was to transfer all 64 disks from one of the three poles to another, with two important constraints. They could only move one disk at a time, and they could never place a larger disk on top of a smaller one. The priests worked very efficiently, day and night, moving one disk every second. When they finished their work, the legend said, the temple would crumble into dust and the world would vanish.



An Example Arrangement of Disks for the Tower of Hanoi

Floodfill





Determinant of a square matrix of order n

$$\det(\mathbf{A}) = \begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{vmatrix}$$

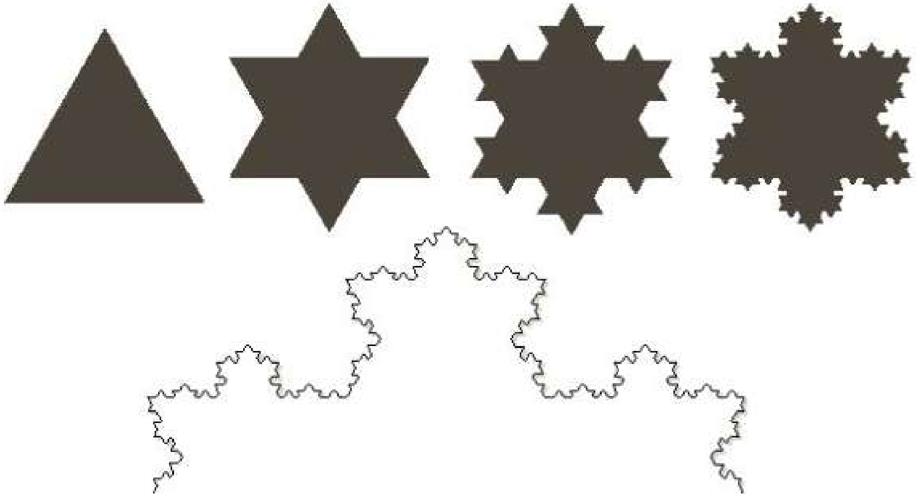
$$det(A) = \sum_{i=1}^{n} (-1)^{i+1} A_{i,1} \det\left(C_{i,1}\right)$$

where $C_{i,1}$ is the $(n-1) \times (n-1)$ matrix obtained from A by removing the i-th row and first column

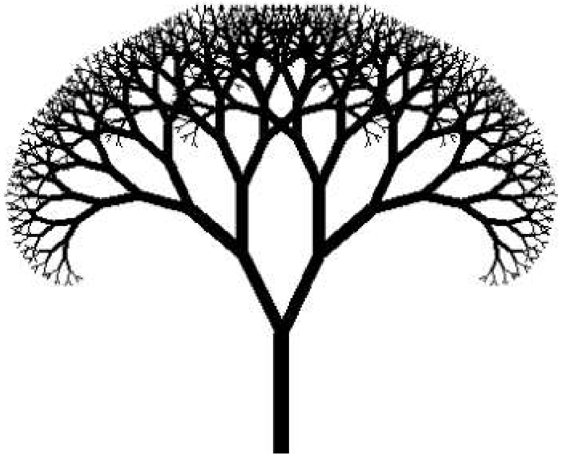
$$\begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{vmatrix}$$

$$\begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{vmatrix}$$

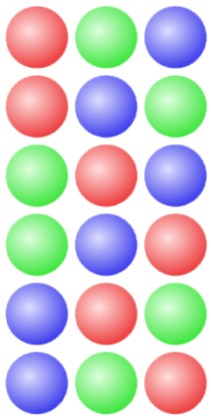
fractals and drawing patterns

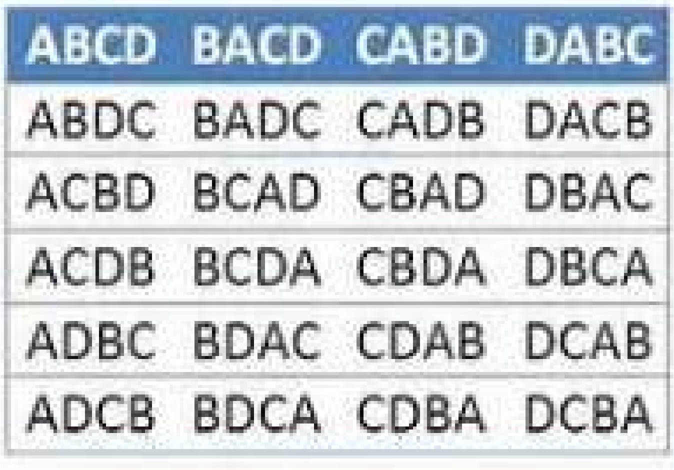


google image search
fractals



Generating permutations

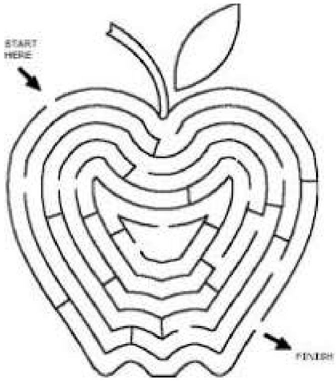


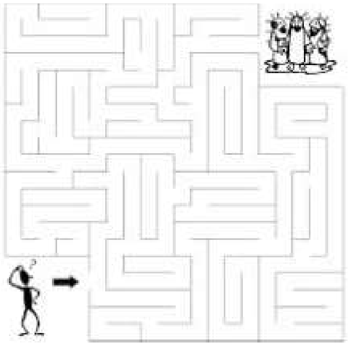


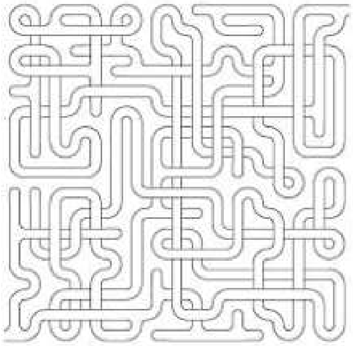
ABCD	BACD	CABD	DABC
ABDC	BADC	CADB	DACB
ACBD	BCAD	CBAD	DBAC
ACDB	BCDA	CBDA	DBCA
ADBC	BDAC	CDAB	DCAB
ADCB	BDCA	CDBA	DCBA

Backtracking

right hand rule







M
A
Z
E

Backtracking

maze

chess: knight tour

chess: 8 / n queen problems

games

traversal of non linear data structures

Miscellaneous

stack frames

tail recursive call optimization

can be easily converted to iterative

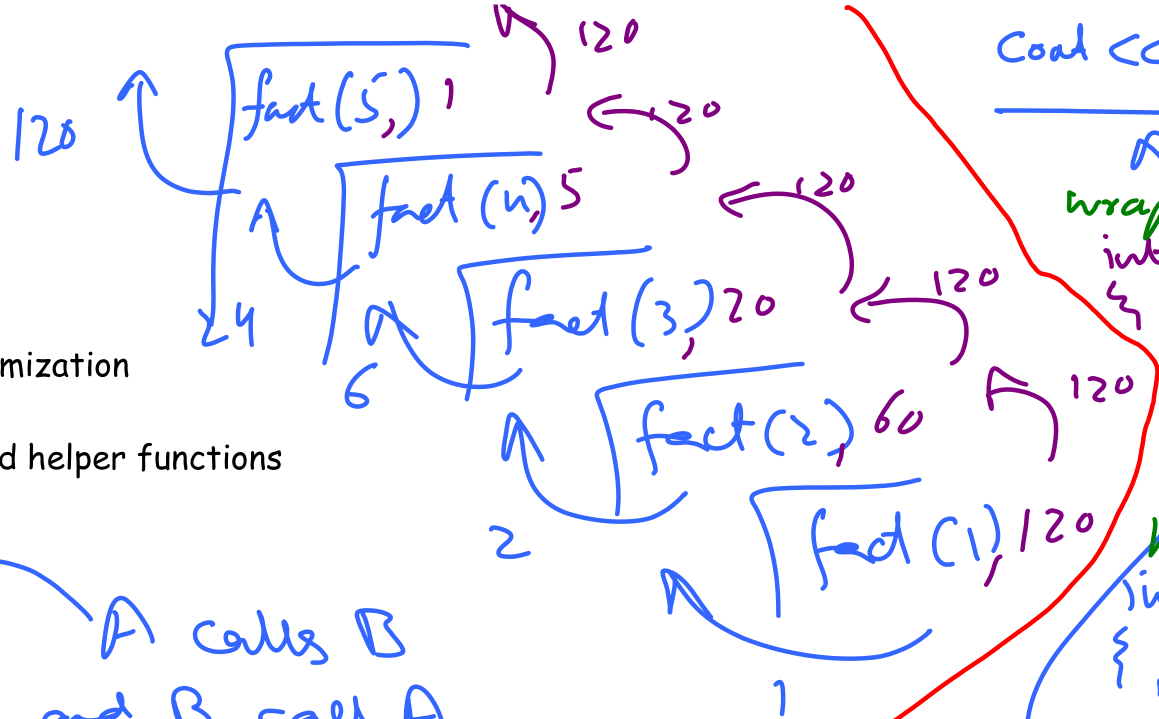
Wrapper functions and helper functions

mutaul recursion

A calls B and B call A

nested recursion

ex: Ackermann function



```
Coat << fact (5,1)
wrapper
int fact(int n)
{
    return fact(n,1);
}
helper
int fact(int n, int r)
{
    if (n == ~~1~~ r)
        return 1
    return fact(n-1, n*r)
}
```

Recursion and iteration

Recursion normaly simple and looks more like the original formulation.

Iteration code will be faster and will use less resources.

(memory)

hybrid approach

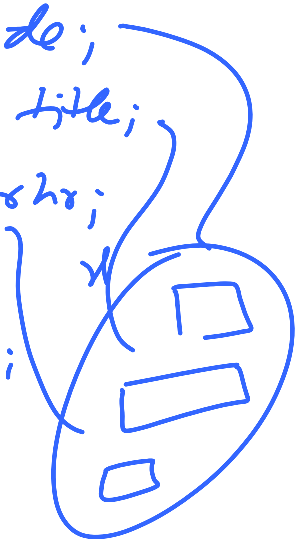
both approaches are used

e.g: to compute determinant use a loop to row 1 and use recusion of smaller determinants.

Recursion in data (structural recursion)

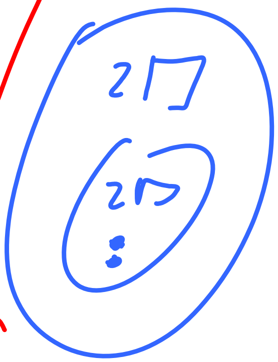
```
struct Course
{
  int code;
  string title;
  int crhr;
};
Course x;
```

✓



```
struct/class XY
{ public:
    int z;
    XY u;
};
XY m;
```

Compiler error.



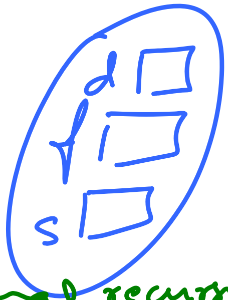
infinite recursion

⟹ But

```
class Node
{
   int d;
   Node *f;
   Node *s;
};
```

is OK

```
Node t;
```



This is structural recursion or recursion in data.

Recursion in data (structural recursion)

```
class Node
{ public:
    int n;
    Node *up;
    Node * left;
    Node *right;
};
```

```
Node first;
first.n = 30;
first.up = new Node;
first.up.up = new Node;
first.up.n = 90;
first.up.left = new Node;
Node two;
two.n = 50;
```

```
first.left = &two;
  ⋮
```

try to understand the provided code.