问题求解论题1-4：算法的基本结构

陶先平 陈道蓄

南京大学

问题1:
你会吃蟹黄汤包吗?

轻轻提，慢慢移，先开窗，再喝汤。

吃一只汤包的“算法”

- 顺序很重要：
  - 将包子从蒸笼中轻轻提起，and then
  - 将包子慢慢移动到面前的碟子中，and then
  - 在包子的上方咬开一个小口，and then
  - 通过小口吸食包子里的汤，and then
  - 将包子送入口中

  完成！

顺序，是组织算法的最基本的方法

按照算法步骤，顺序书写指令

如何理解下面这句话？

- My second remark is that our intellectual powers are rather geared to master static relations and that our powers to visualize processes evolving in time are relatively poorly developed. For that reason we should do our utmost to shorten the conceptual gap between the static program and the dynamic process, to make the correspondence between the program (spread out in text space) and the process (spread out in time) as trivial as possible.

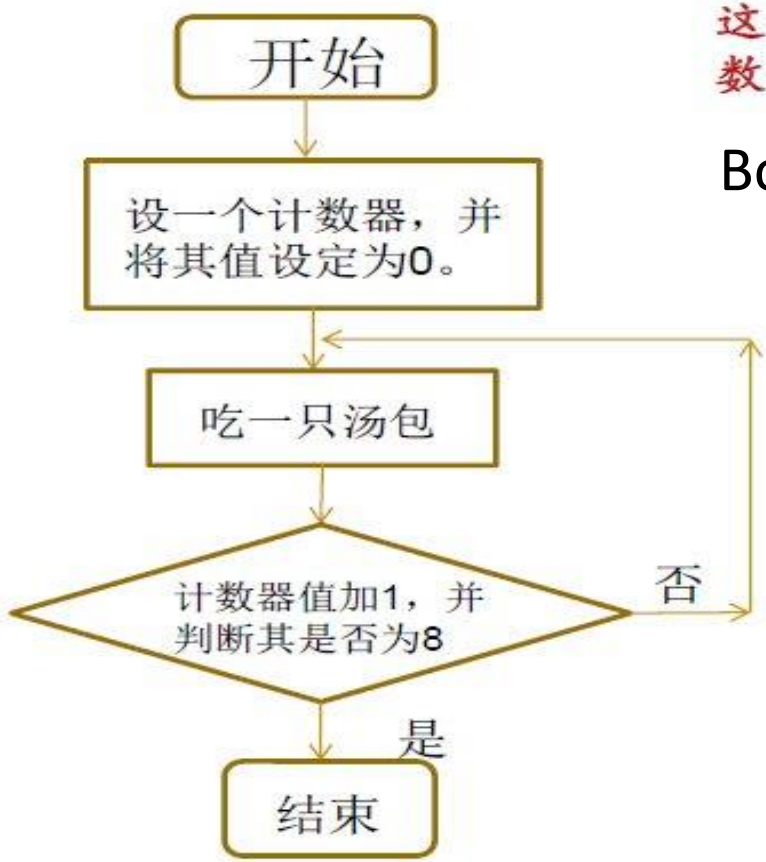
----E.W.Dijkstra

问题2：

但是我们并不只吃一只，怎么办？

策略一：控制数量

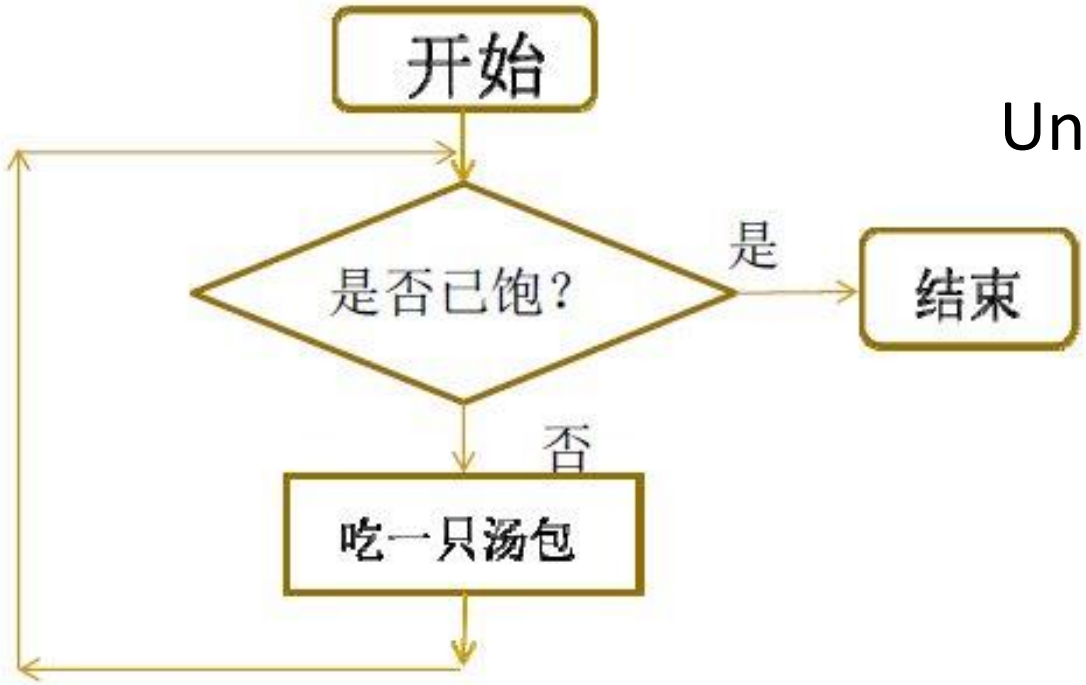
假如规定吃8只：



注意：
这个过程的“结构”与计数器的初始值没有关系！

Bound iteration

策略二：吃饱为止

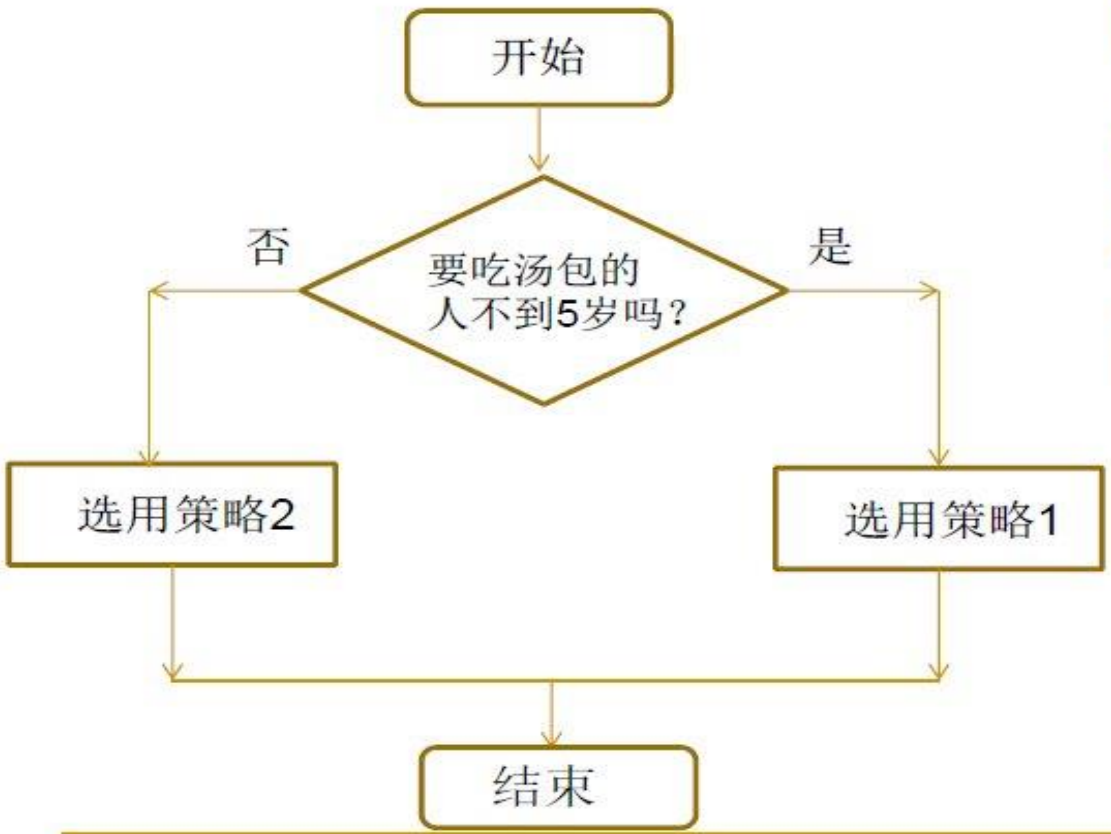


Unbound iteration

如果即使饱了，也希望再品尝一个，该怎么办？

问题：你能正确选择采用何种策略设计循环吗？

有人知道饱不饱，但有人不知道！



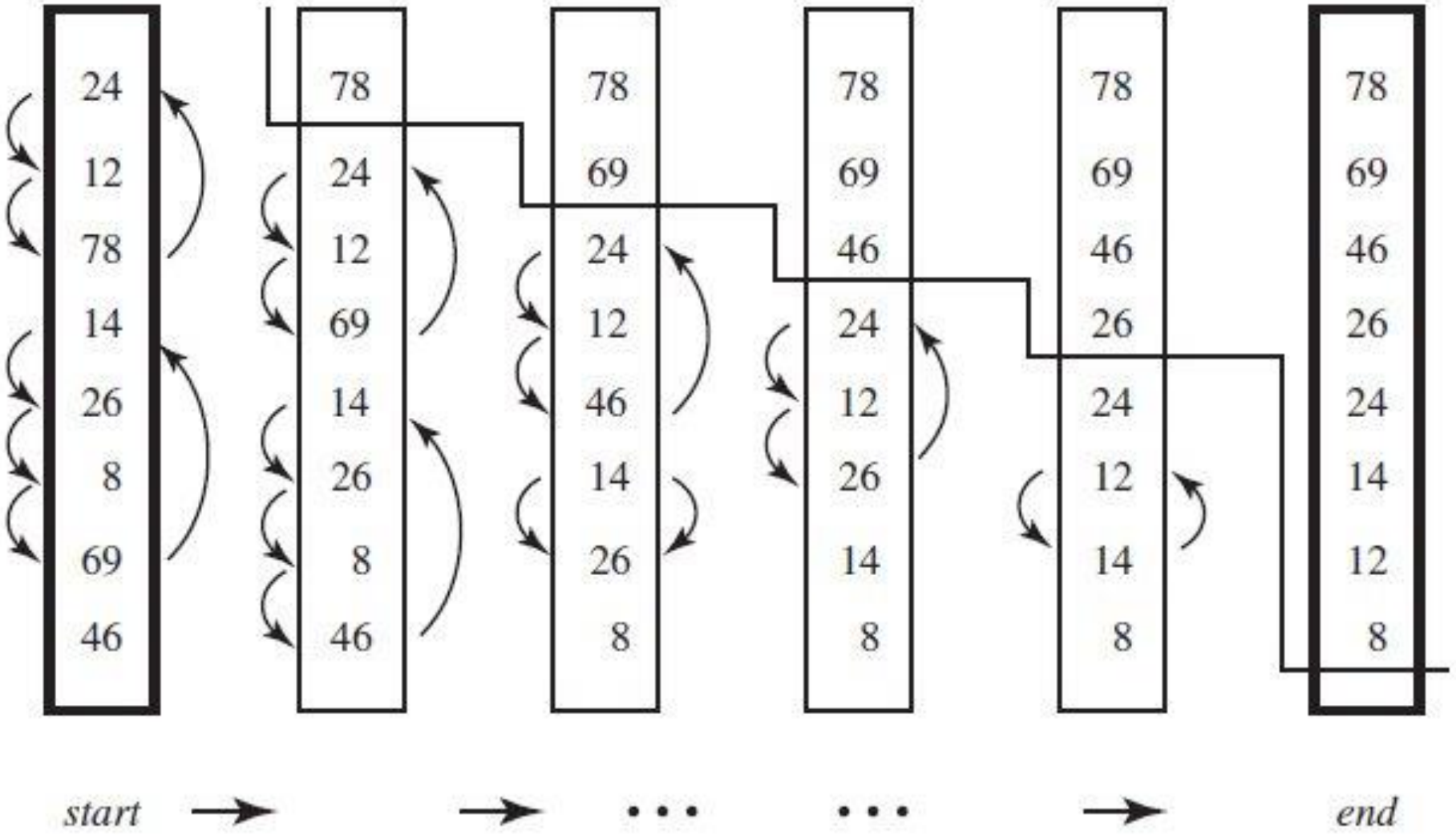
问题7：

如果要判断的不止是两种可能，那怎么办？

问题：

如果说顺序、分支和循环构成了算法组织的三个基本结构，那么顺序结构是算法组织的基本中的基本。为什么？

嵌套结构：循环嵌套


24
12
78
14
26
8
69
46
78
24
12
69
14
26
8
46
78
69
24
12
46
14
26
8
78
69
46
24
12
26
14
8
78
69
46
26
24
12
14
8
78
69
46
26
24
14
12
8
start
end

冒泡排序的算法描述

(1) do the following $N - 1$ times:
    (1.1) point to the first element;
    (1.2) do the following $N - 1$ times:
        (1.2.1) compare the element pointed to with the next element;
        (1.2.2) if the compared elements are in the wrong order, exchange them;
        (1.2.3) point to the next element.

如何确定循环过程是正确的？

- 循环不变式(循环不变量)：
  - 一个逻辑表达式
    - 该表达式通常可用来表征循环的功能
  - 循环初试时为真
  - 在循环执行过程中始终为“真”
  - 循环结束时，必定为真

下例的循环不变式是什么？

QUOTE:

```
Apvector <int>  list(n);     // n is some positive integer
int k, indexMax;
<code which assigns values to all entries in list>
indexMax = 0;
for(k = 1; k < list.length(); k++){
        // invariant true here
        if (list[k] > list[indexMax])
        indexMax = k;
}
```

冒泡排序的循环不变量是什么？

(1) do the following $N - 1$ times:
    (1.1) point to the first element;
    (1.2) do the following $N - 1$ times:
        (1.2.1) compare the element pointed to with the next element;
        (1.2.2) if the compared elements are in the wrong order, exchange them;
        (1.2.3) point to the next element.

关于Goto语句

```
#include"stdio.h"
        int main(void){
            int n=0;
            printf("input a string ：  \n");
    loop: if(getchar()!='\n') {
            n++;
            goto loop;
            }
            printf("%d",n);
        }
```

Goto语句

```
#include <stdio.h>

int main(void){
        int i, j, k;

        for (i = 0; i < 10; i++)
            for (j = 0; j < 10; j++){
                for (k = 0; k < 10; k++)
                    if (i + j + k = 10)
                    break;
                printf("%d %d %d", i, j, k);
            }
}
```

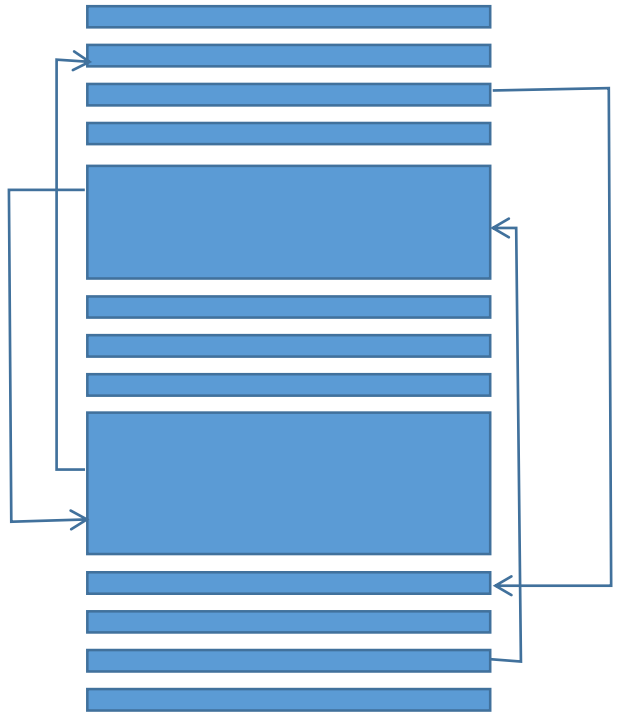
```
#include <stdio.h>

int main(void){
        int i, j, k;

        for (i = 0; i < 10; i++)
            for (j = 0; j < 10; j++){
                for (k = 0; k < 10; k++)
                    if (i + j + k = 10)
                    goto exit_for;
            }
exit_for:   printf("%d %d %d", i, j, k);
}
```

如果不加限制地使用goto语句：

- 面条式的程序：
  - 文本空间中的程序：
    - 下锅前的面条
  - 运行中的执行序列：
    - 下锅后的面条
- 难以想象！
  - 文本空间和时间空间上巨大的gap



重新阅读下面这句话？

- My second remark is that our intellectual powers are rather geared to master static relations and that our powers to visualize processes evolving in time are relatively poorly developed. For that reason we should do our utmost to shorten the conceptual gap between the static program and the dynamic process, to make the correspondence between the program (spread out in text space) and the process (spread out in time) as trivial as possible.

----E.W.Dijkstra

顺序结构、分支结构、循环结构这三种结构产生的gap？
Goto结构产生的gap？

理解并记住：

clear and well structured. Clarity and structure, as is repeatedly emphasized, are of the utmost importance in algorithmics, and many an effort is devoted to finding ways of imposing them on algorithm designers.

Why？

何为：控制结构的表达能力

- Various minimal sets of control structures have been identified, meaning that in certain technical senses other control structures can be replaced by appropriate combinations of those in the minimal set, so that in practice these are the only ones needed.

However, using this result to rid a given algorithm of its subroutines involves adding considerable "machinery" (in the form of new elementary instructions) to the algorithm.

大师与“go to语句”

Edsjer Wybe Dijkstra (1930-2002), 获1972年图林奖

For fundamental contributions to programming as a high, intellectual challenge; for eloquent insistence and practical demonstration that programs should be composed correctly, not just debugged into correctness; for illuminating perception of problems at the foundations of program design.

Dijkstra, Edsger W., “Go To statement considered harmful,” Communications of the ACM, Vol. 11, Num. 3, 1968, pp. 147–148. The Letter to the Editor that ignited the infamous “Go To” controversy. Submitted under the title “A case against the GO TO statement,” and retitled by the editor. The title template “X considered harmful” was for many years used in the titles of rants and diatribes.

关于“Goto”的历史逸事

The title of Edsger Dijkstra's 1968 "Go To Statement Considered Harmful" is among the best-known phrases in the history of programming. Interestingly, the phrasing of the title — which has become so regular a cliche in the field it inspired Eric Meyer to compose the waggish "'Considered Harmful' Essays Considered Harmful" — was not Dijkstra's work at all. As Dijkstra explained it:

. In 1968, the Communications of the ACM published a text of mine under the title "The goto statement considered harmful," which in later years would be most frequently referenced, regrettably, however, often by authors who had seen no more of it than its title, which became a cornerstone of my fame by becoming a template: we would see all sorts of articles under the title "X considered harmful" for almost any X, including one titled "Dijkstra considered harmful." But what had happened? I had submitted a paper under the title "A case against the goto statement", which, in order to speed up its publication, the editor had changed into a "letter to the Editor", and in the process he had given it a new title of his own invention! The editor was Niklaus Wirth.

一种新的算法组织方法：子程序(过程,函数)

- 问题：
  - 什么情况下，我们会想到用过程来“封装”一段算法？

- 问题：
  - 过程通常都有一种叫“形式参数”的东西，你能从“过程的参数”中想到什么？

一种新的算法组织方法：子程序(过程,函数)

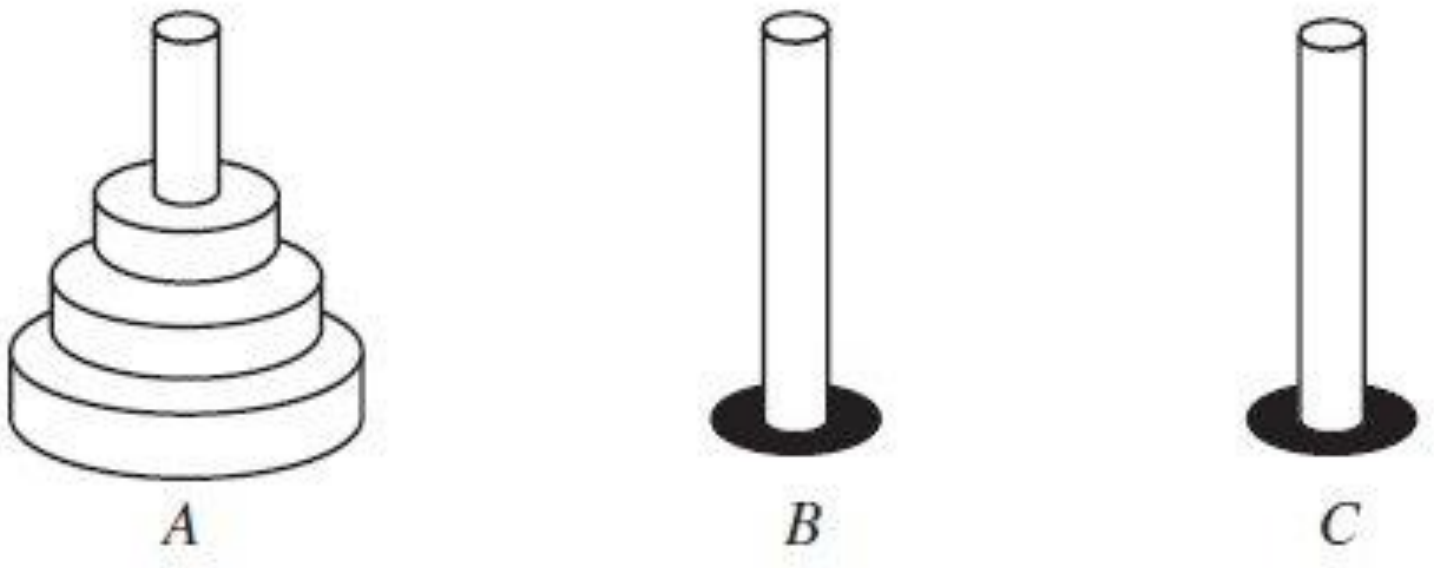
- 过程的优点：
  - 复用
  - 封装
  - 抽象

- 丰富了算法结构：
  - 顺序、
  - 分支、循环
  - 调用

问题

什么是Top down 或者 Bottom up算法设计方法？

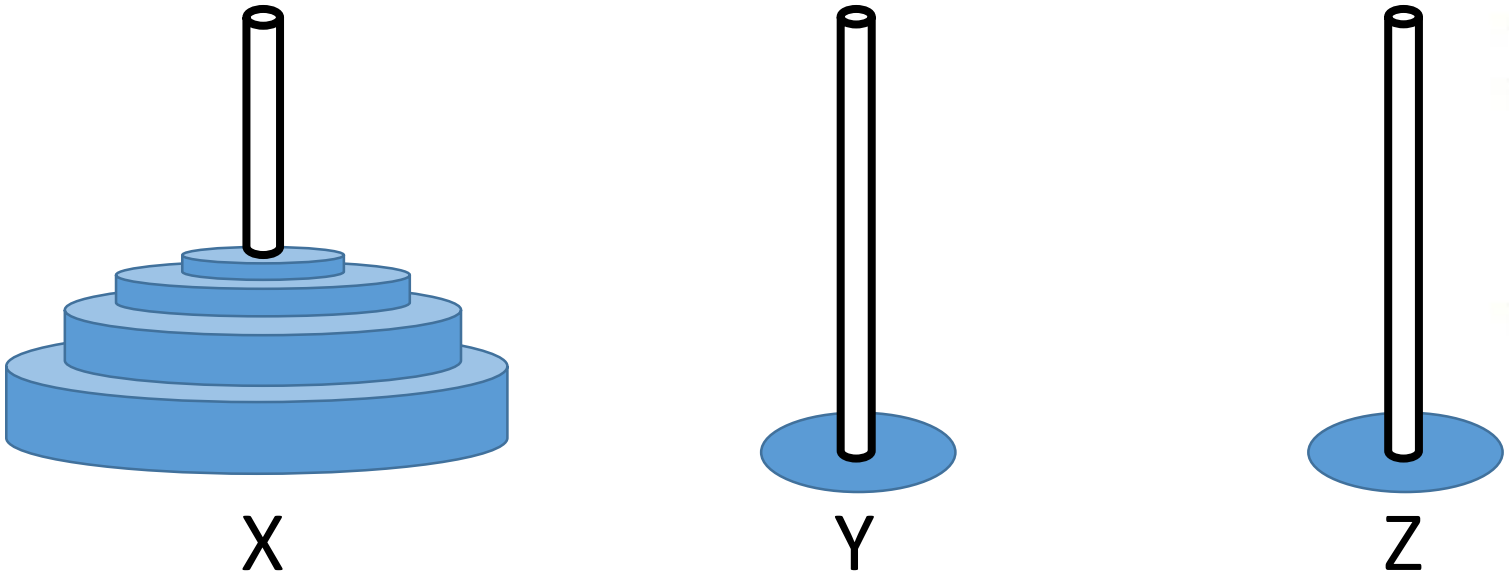
递归：自己调用自己的过程



它怎么就递归了呢？

问题：它怎么就递归了呢？

- 定义Move N from X to Y using Z
  - 当N=4时



subroutine **move** N **from** X **to** Y **using** Z:

(1) if N is 1 then output "move X to Y";

(2) otherwise (i.e., if N is greater than 1) do the following:
  (2.1) call **move** $N-1$ **from** X **to** Z **using** Y;
  (2.2) output "move X to Y";
  (2.3) call **move** $N-1$ **from** Z **to** Y **using** X;

(3) return.

递归：自己调用自己的过程

- 例题：从一个很长的数列中，怎样找到最大的数？

问题：为什么这个问题能够用递归的方法去做？

观察：

递归调用自身时，输入的参数相比前一次调用时使用的参数，在值上通常会有什么特点？

思考：递归方法和数学归纳法有关联关系吗？