



《计算概论A》课程 程序设计部分

函数的递归调用 (1)

李 戈

北京大学 信息科学技术学院 软件研究所

2010年11月30日



北京大学

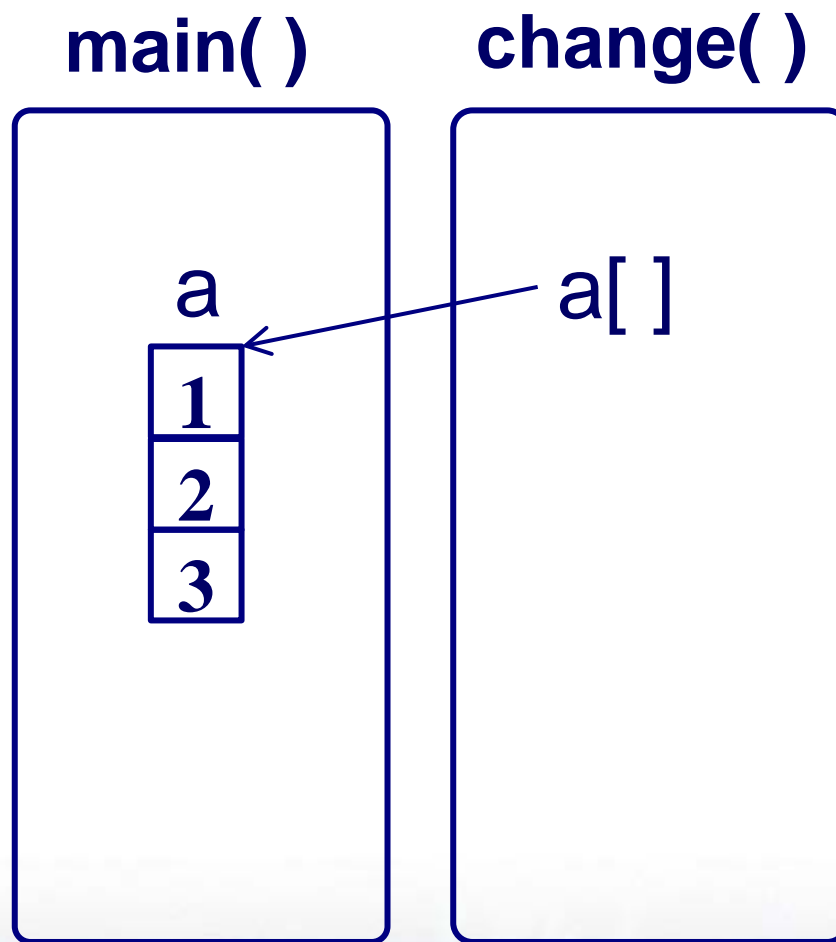
复习题 (1)

```
#include<iostream>
using namespace std;
void change(int [] *a)
{
    a[0] = 0;
    a[1] = 0;
    a[2] = 0;
}
int main()
{
    int a[3][4]={1, 2, 3};
    change(a);
    cout<<a[0]<<a[1]<<a[2]<<endl;
    return 0;
}
```



复习题 (1)

```
#include<iostream>
using namespace std;
void change(int a[ ])
{
    a[0] = 0;
    a[1] = 0;
    a[2] = 0;
}
int main()
{
    int a[3]={1, 2, 3};
    change(a);
    cout<<a[0]<<a[1]
        <<a[2]<<endl;
    return 0;
}
```





复习题 (2)

```
struct stru
{
    int n;
    char c;
};
void change(stru b)
{ b.n=20;  b.c='y';}
int main()
{
    stru a = {10,'x'};
    change(a);
    cout<<a.n<<" "<<a.c;
    return 0;
}
```

main()

a	
n	10
c	x

change()

b	
n	20
c	y



北京大学

复习题 (2)

```
#include<iostream>
using namespace std;
struct stru
{
    int n;
    char c;
};
stru change(stru b)
{
    b.n=20;    b.c='y';
    return b;
}
int main()
{
    stru a = {10,'x'};
    a = change(a);
    cout<<a.n<<" "<<a.c;
    return 0;
}
```

main()

a	
n	10
c	x

change()

b	
n	20
c	y



北京大学



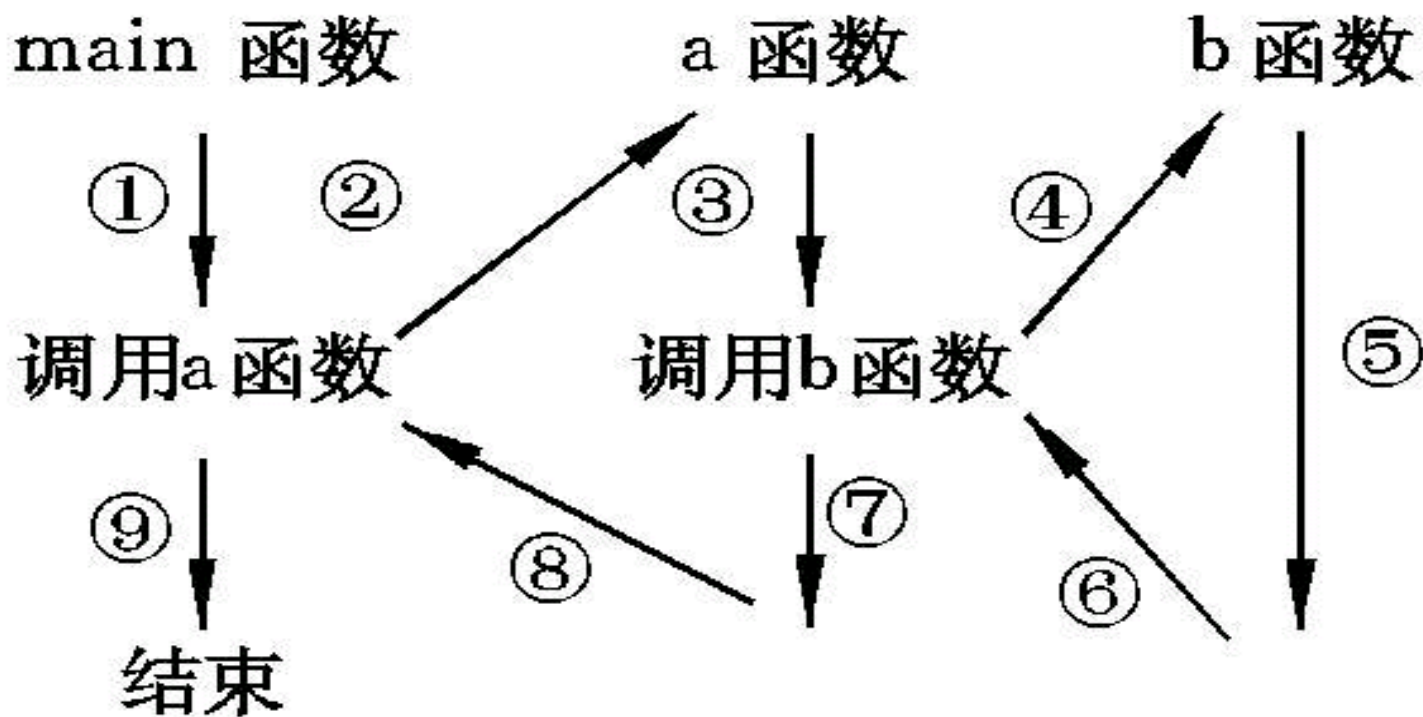
从 嵌套 到 递归



清华大学

函数的嵌套调用

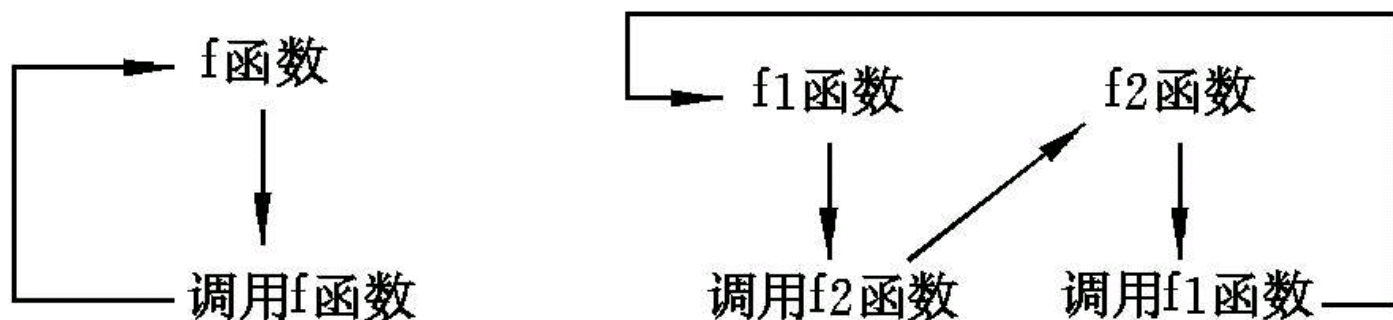
- 函数不能嵌套定义，但可以嵌套调用
 - ◆ 在调用一个函数的过程中，又调用另一函数



函数的递归调用

■ C++语言允许递归调用

- ◆ 在调用一个函数的过程中又出现直接或间接地调用该函数本身。



- ◆ 程序中不应出现无终止的递归调用，必须控制只有在某一条件成立时才继续执行递归调用，否则就不再继续。





例题分析 (1)

- 有5个人坐在一起，问第5个人多少岁？他说比第4个人大2岁。问第4个人岁数，他说比第3个人大2岁。问第3个人，又说比第2个人大2岁。问第2个人，说比第1个人大2岁。最后问第1个人，他说是10岁。请问第5个人多大。

- ◆ $\text{Age}[1] = 10;$
- ◆ $\text{Age}[2] = \text{age}[1] + 2;$
- ◆ $\text{Age}[3] = \text{age}[2] + 2;$
- ◆ $\text{Age}[4] = \text{age}[3] + 2;$
- ◆ $\text{Age}[5] = \text{age}[4] + 2;$





例题分析 (1)

```
#include <iostream.h>
```

```
int main()
```

```
{ int age[6];
```

```
    age[1] = 10;
```

```
    for (int i = 2; i <= 5; i++)
```

```
    { age[i] = age[i-1] + 2;}
```

```
    cout << "第5个人的年龄是: " << age[5] << endl;
```

```
    return 0;
```

```
}
```



北京大學

另一种解决方案

```
#include<iostream.h>
```

```
int age(int n)
```

```
{ int c;
```

```
    if(n == 1)  
        c = 10;
```

```
    else
```

```
        c = age(n-1)+2;
```

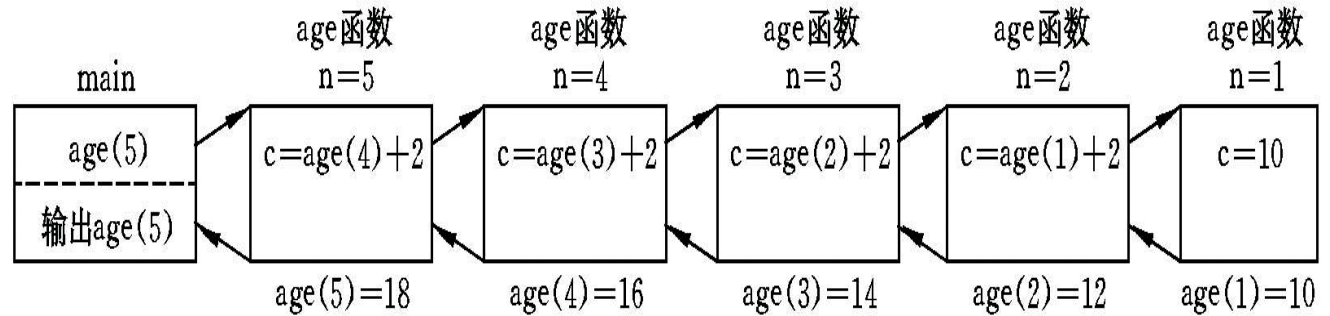
```
    return(c);
```

```
}
```

```
void main()
```

```
{ cout<<“第五个人的年龄是: ” <<age(5);
```

```
}
```



北京大学



另一种解决方案

```
#include<iostream.h>
```

```
int age(int n)
```

```
{ int c;
```

```
  if(n == 1)
```

```
    c = 10;
```

```
  else
```

```
    c = age(n-1)+2;
```

```
  return(c);
```

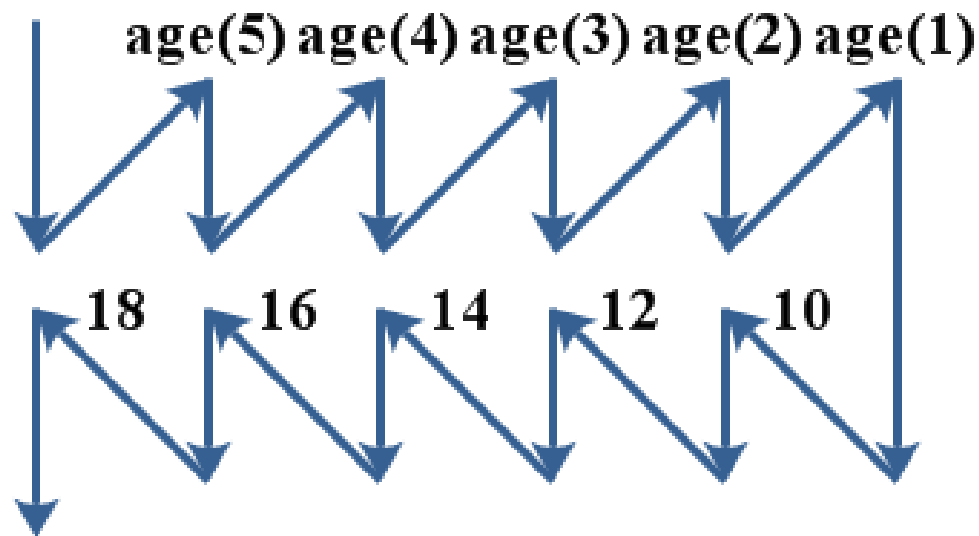
```
}
```

```
void main()
```

```
{ cout<<“第五个人的年龄是: ” <<age(5);
```

```
}
```

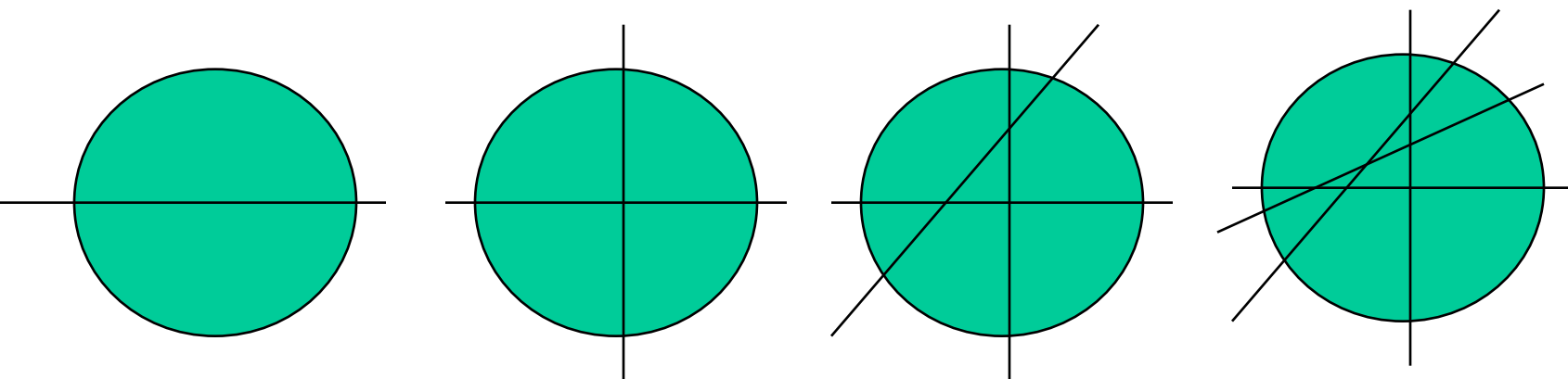
Main()



北京大学

例题分析 (2)

■ 切饼，100刀最多能切多少块？



- $q(1) = 1 + 1 = 2$

- $q(2) = 1 + 1 + 2 = 4;$

- $q(3) = 1 + 1 + 2 + 3 = 7;$

- $q(4) = 1 + 1 + 2 + 3 + 4 = 11;$

- $q(n) = q(n-1) + n; q(0) = 1;$



北京大學



例题分析 (2)

```
#include <iostream.h>
```

```
int main()
```

```
{  int q[101];
```

```
    q[0] = 1;
```

```
    for (int i = 1; i <= 100; i++)
```

```
    {  q[i] = q[i-1] + i;}
```

```
    cout<<"100刀最多可切"<<q[100]<<"块"<<endl;
```

```
    return 0;
```

```
}
```

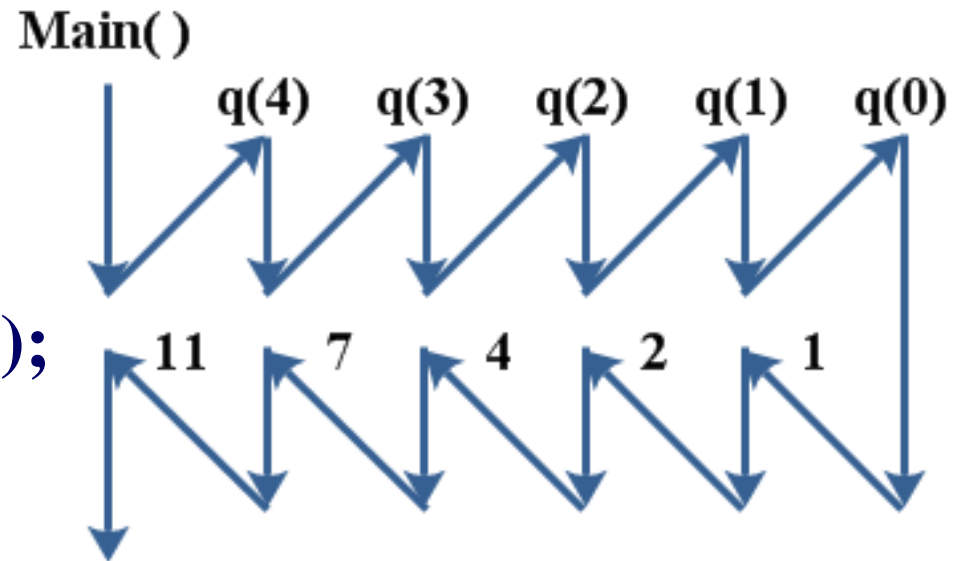


北京大学

另一种解决方案

```
#include<iostream>
using namespace std;
int q(int n){
    if (n == 0)
        return 1;
    else
        return( n + q(n-1));
}
```

```
int main(){
    cout<<q(4)<<endl;
    return 0;
}
```





递推数列

- 一个数列从某一项起，它的任何一项都可以用它前面的若干项来确定，这样的数列称为递推数列：
 - ◆ $1!, 2!, 3!, \dots n!$
 - $\text{fact}(n) = n * \text{fact}(n-1)$ （通项公式）；
 - $\text{fact}(1) = 1$ （边界条件）
 - ◆ $1, 1, 2, 3, 5, 8, 13, 21, \dots$
 - $\text{fab}(n) = \text{fab}(n-1) + \text{fab}(n-2)$ （通项公式）；
 - $\text{fab}(1) = 1, \text{fab}(2) = 1$ ；（边界条件）





递推与递归

■ 递推问题

◆ 后续的运算依赖于已知的条件；当前的运算是下一步运算的基础；

■ 解法1：从已知的初始条件出发，逐次去求所需要的阶乘值。

如求 $n!$

初始条件 $\text{fact}(1) = 1$

$$\text{fact}(2) = 2 * \text{fact}(1) = 2$$

$$\text{fact}(3) = 3 * \text{fact}(2) = 6$$

... ..



北京大學



递推与递归

■ 递推问题

- ◆ 后续的运算依赖于已知的条件；当前的运算是下一步运算的基础；

■ 解法2：递归算法

- ◆ 出发点不放在初始条件上，而放在求解的目标上，从所求的未知项出发逐次调用本身的求解过程，直到递归的边界（即初始条件）。



例题分析 (3)

■ 问题：已知 n ，求 $n!$

假设计算阶乘的任务由一个函数 fact 来做

$\text{fact}(n)$ 等于 $\text{fact}(n-1)*n$

$\text{fact}(n-1)$ 等于 $\text{fact}(n-2)*(n-1)$

$\text{fact}(3)$ 等于 $\text{fact}(2)*3;$

$\text{fact}(2)$ 等于 $\text{fact}(1)*2$

$\text{fact}(1)$ 等于 $1;$

■ 可知

◆ $\text{fact}(n)$ 的值等于 $\text{fact}(n-1)*n;$

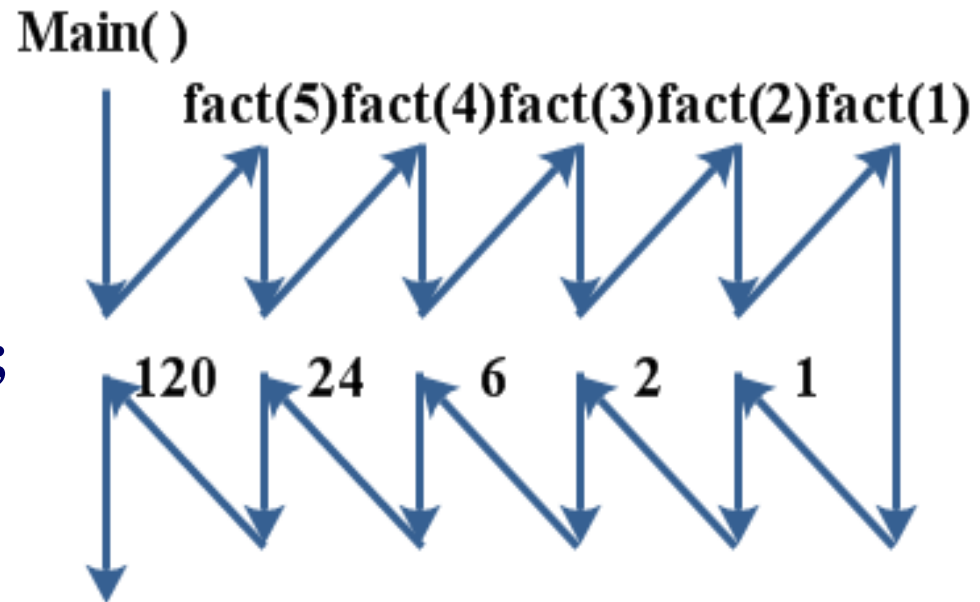
◆ $\text{fact}(1) = 1;$



北京大學

例题分析 (3)

```
#include<iostream>
using namespace std;
int Factorial(int n){
    int temp;
    if(n == 1)
        return 1;
    else{
        temp = Factorial( n-1 );
        temp = n*temp;
        return temp;
    }
}
int main(){
    cout<<Factorial(5)<<endl;
    return 0;
}
```



例题分析 (3)

栈与递归

F=Factorial(5)-请输入n的值

Cooling



Clear Restart

(1) int Factorial(int n);

(2) { int temp;

(3) if(n==0)
演示结束, 要重来请按
Restart键!

(5) else{

(6) temp=Factorial(n-1);

(7) temp=n*temp; 知道了

(8) return temp;

(9) }

(10) }

Factorial(5)=120

参数表

返回地址

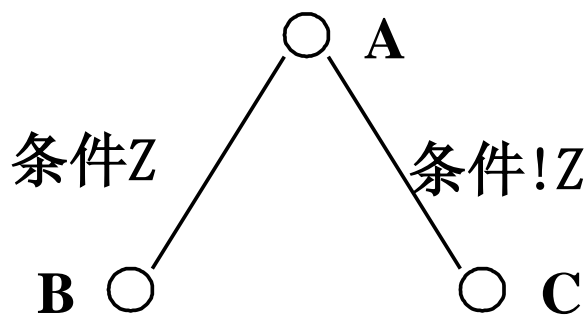
Top





为了表述得直观清晰，我们定义一种图来辅助表示算法，该图两个结点：或结点与与结点。

1、或结点



$$A = \begin{cases} B, Z = true \text{ (真)} \\ C, Z = false \text{ (假)} \end{cases}$$

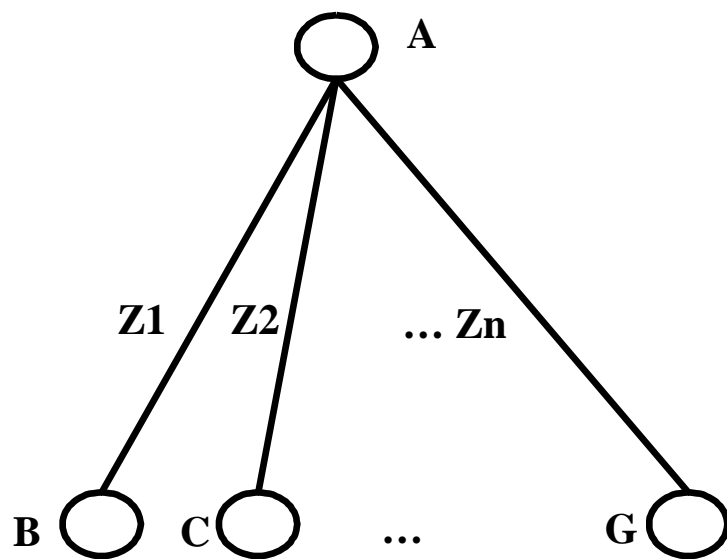
A为“或结点”，A依不同条件会有两种不同的取值B或C。结点用 O 表示。



北京大学



如果有多于2种取值，可用下图：



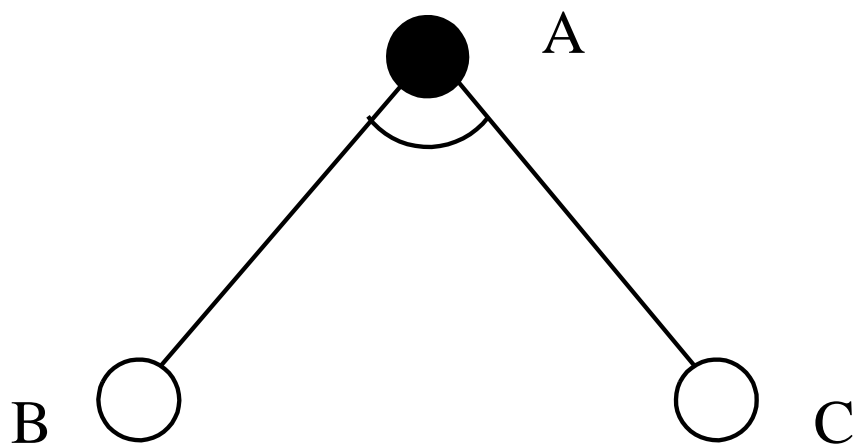
条件为 Z_1 , Z_2 , ..., Z_n , 取值为B或C, ...或G



北京大学



2、与结点



与结点要涂黑，相关联的B与C之间要用弧线连起来。

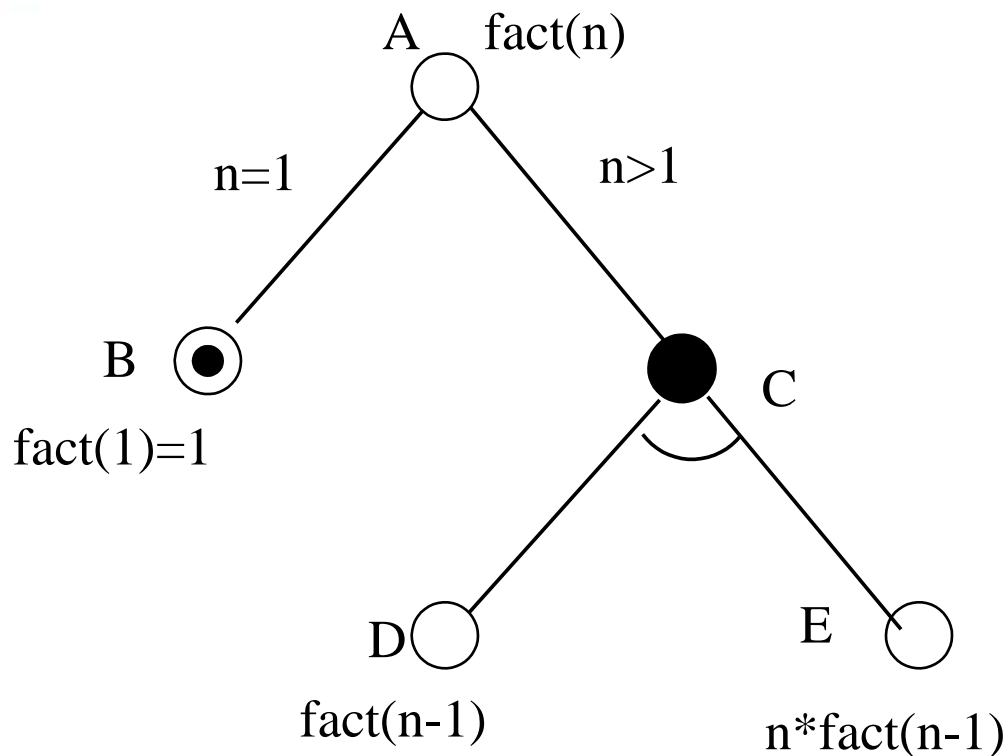
A为与结点，A的最终取值为C结点的值，但为了求得C的值，得先求出B结点的值，C是B的函数。



北京大学



以求 $n!$ 为例画出如下与或图

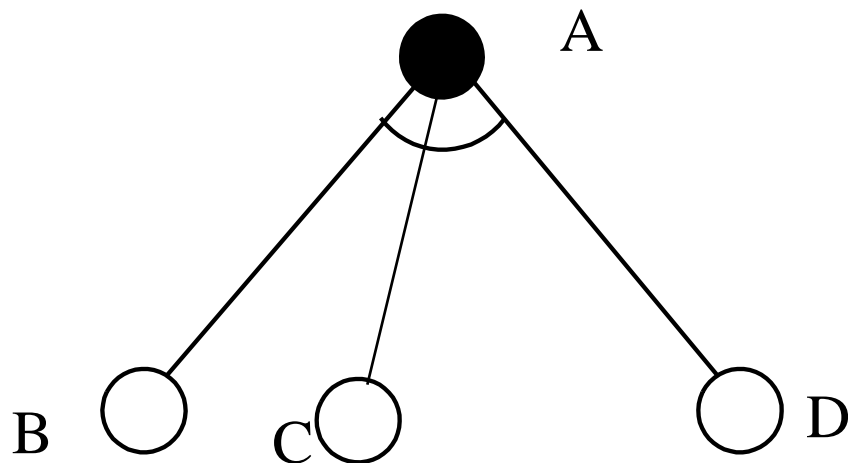


- A为或结点；B为直接可解结点，值为1；
- C为与结点，当 $n>1$ 时，A的取值即C的值，而C的值即E的值，为了求得E的值，需要先求出D的值。D值 $\text{fact}(n-1)$ 乘以 n 即为E的值。





与结点可能有多个相关联的点，这时可描述为下图

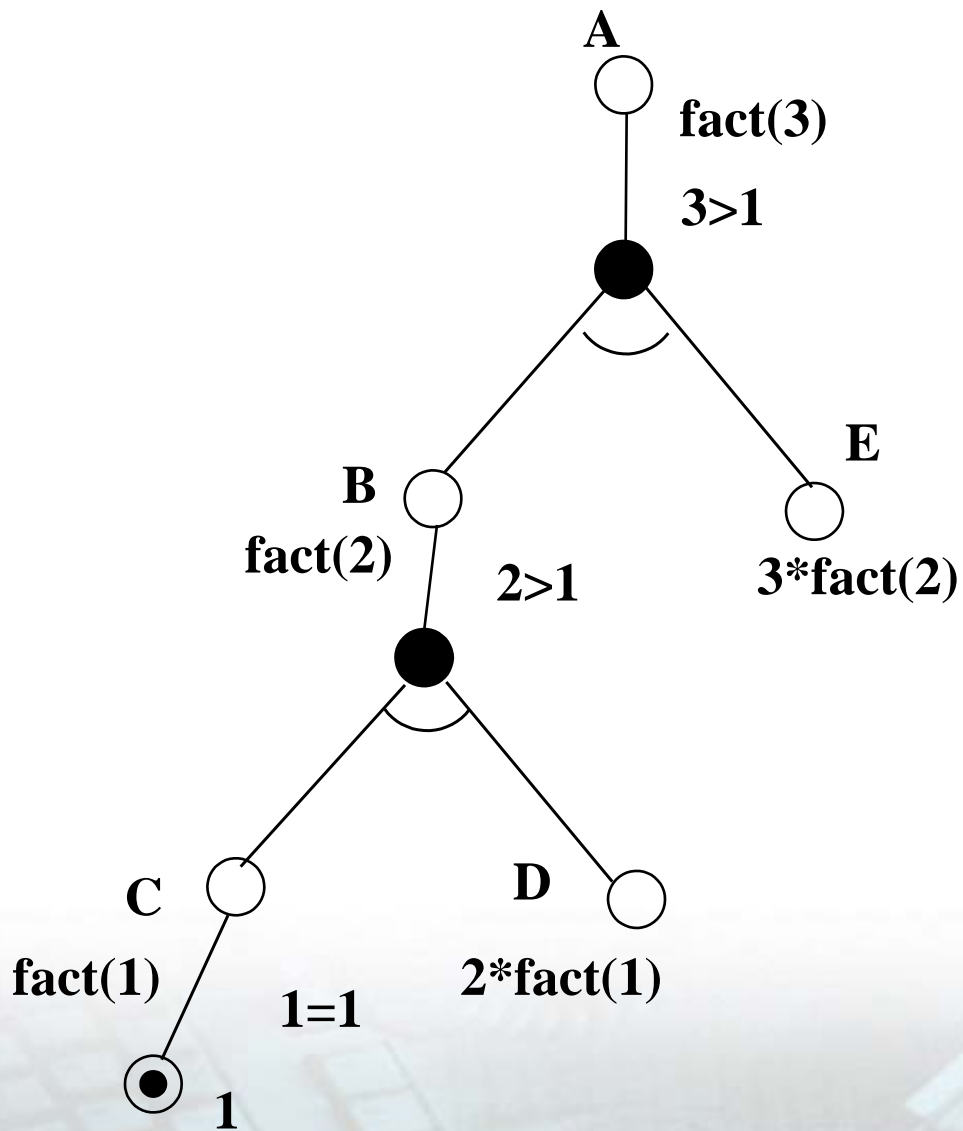


- A结点的值最终为D的值，但为了求D需先求B和C。从图上看先求左边的点才能求最右边的点的值，我们约定最右边D点的值就是A结点的值。



北京大学

- 以3! 为例来画与或图



$$A = E = 6$$

$$E = 3*B = 3*2 = 6$$

$$B = D = 2$$

$$D = 2*C = 2$$

$$C = 1$$





递归问题例题



北京大学



程序分析 (1)

```
#include<iostream.h>
void function(int num){
    cout<<num%10;
    if ((num/10)!= 0)
        function (num/10);
}
void main(){
    int num;
    cin>>num;
    function (num);
}
```

试分析这个程序执行了什么功能。



北京大学



程序分析 (2)

```
#include<iostream>
using namespace std;
void echo()
{ char c;
  c = cin.get();
  if (c != '\n')  echo( );
  cout<<c;
}
void main()
{ echo( ); }
```

试分析这个程序执行了什么功能。

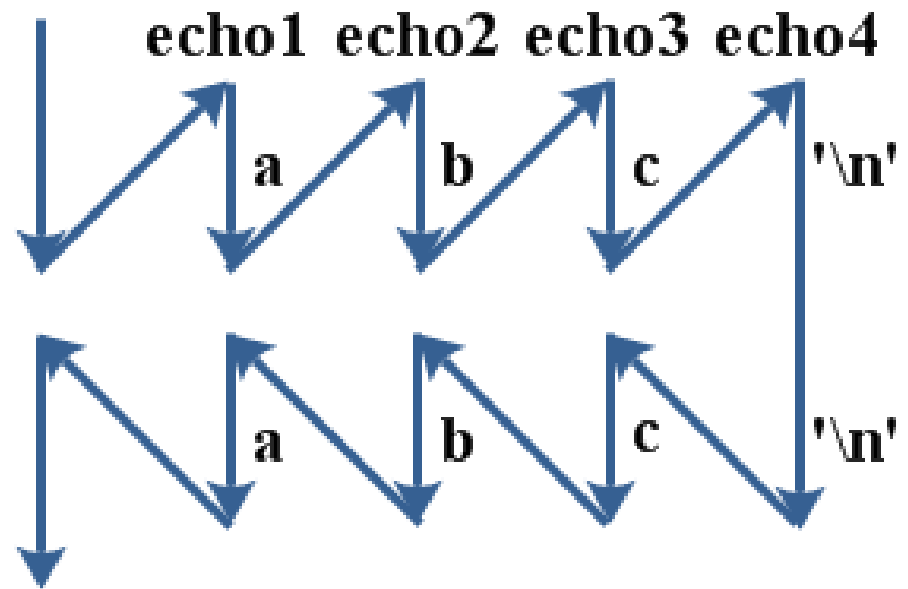


北京大学

程序分析 (2)

```
#include<iostream>
using namespace std;
void echo()
{ char c;
  c = cin.get();
  if (c != '\n')  echo( );
  cout<<c;
}
void main()
{ echo( ); }
```

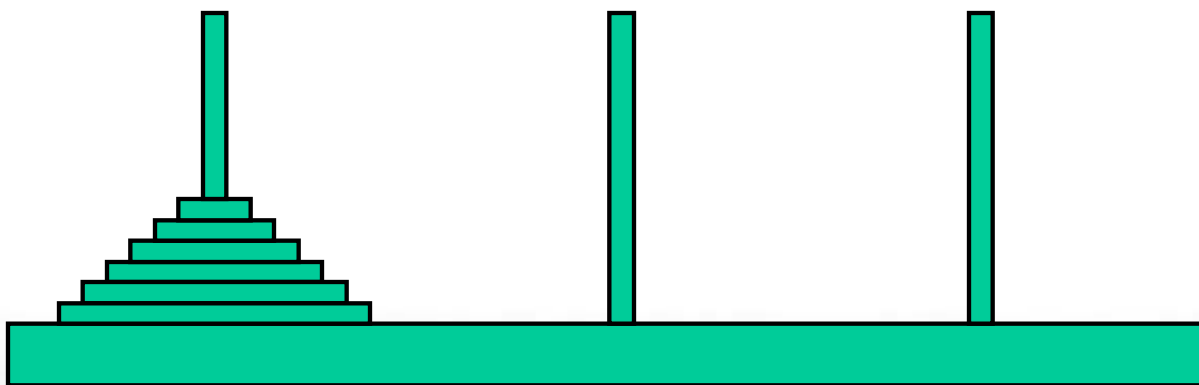
Main()



北京大学

递归经典问题——汉诺塔问题

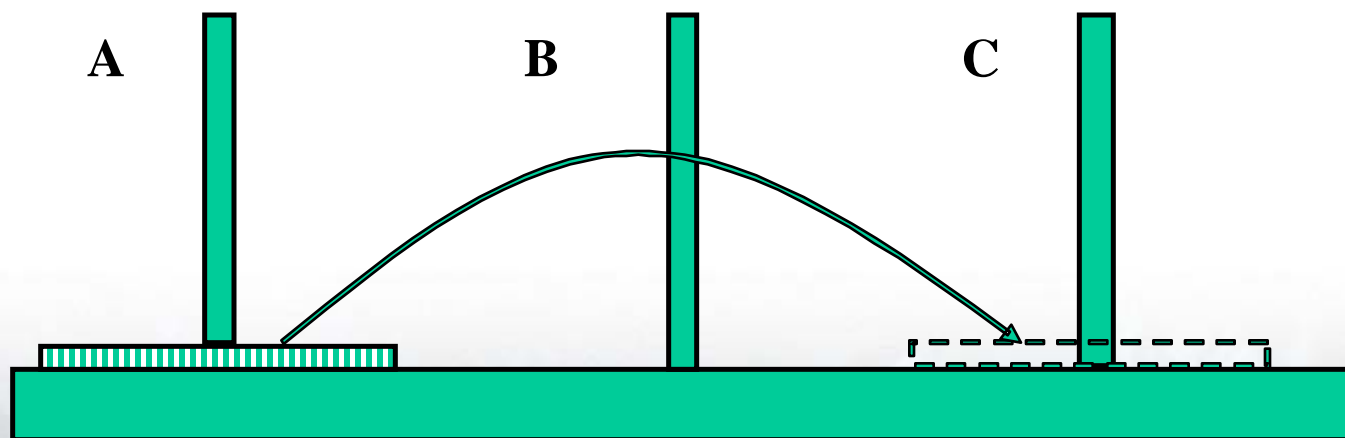
- 故事：相传在古代印度的Bramah庙中，有位僧人整天把三根柱子上的金盘倒来倒去，原来他是想把64个一个比一个小的金盘从一根柱子上移到另一根柱子上去。移动过程中恪守下述规则：每次只允许移动一只盘，且大盘不得落在小盘上面。
- 有人会觉得这很简单，真的动手移盘就会发现，如以每秒移动一只盘子的话，按照上述规则将64只盘子从一个柱子移至另一个柱子上，所需时间约为5800亿年。



递归经典问题——汉诺塔问题

怎样编写这种程序？从思路上还是先从最简单的情况分析起，搬一搬看，慢慢理出思路。

- 1、在A柱上只有一只盘子，假定盘号为1，这时只需将该盘从A搬至C，一次完成，记为move 1 from A to C



北京大学

2、在A柱上有二只盘子，1为小盘，2为大盘。

第（1）步将1号盘从A移至B；

第（2）步将2号盘从A移至C；

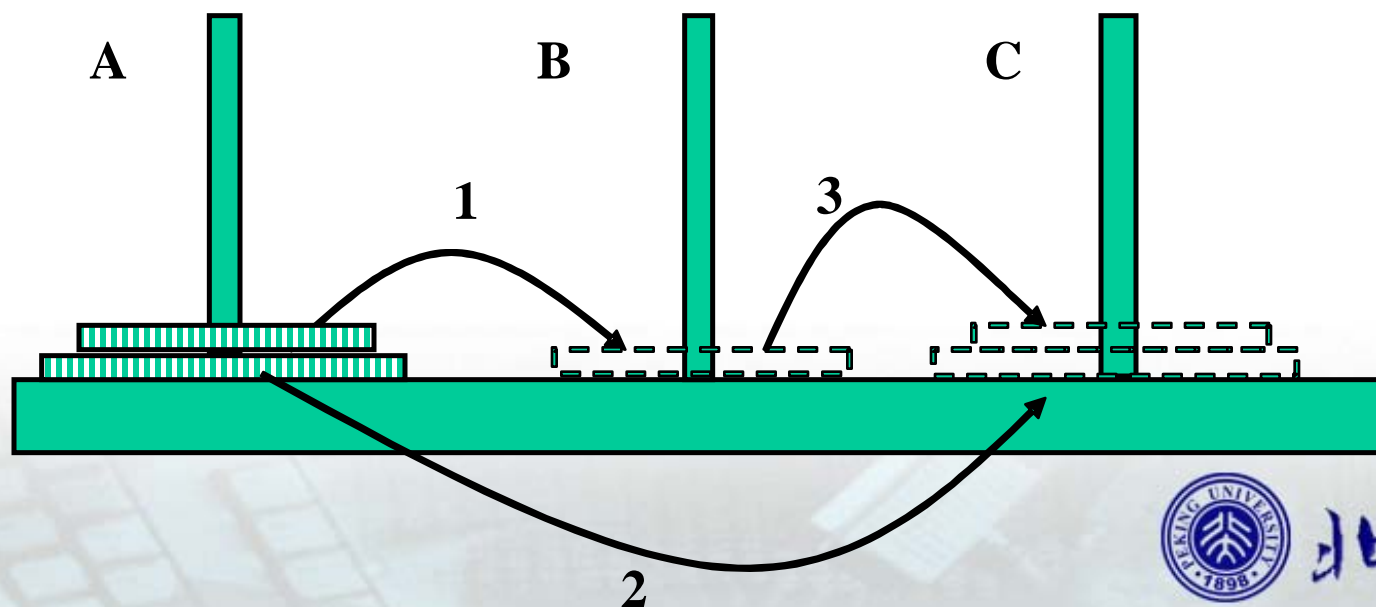
第（3）步再将1号盘从B移至C；

这三步记为：

move 1 from A to B;

move 2 from A to C;

move 3 from B to C;



北京大學

3、在A柱上有3只盘子，从小到大分别为1号，2号，3号

第（1）步将1号盘和2号盘视为一个整体；先将二者作为整体从A移至B。这一步记为

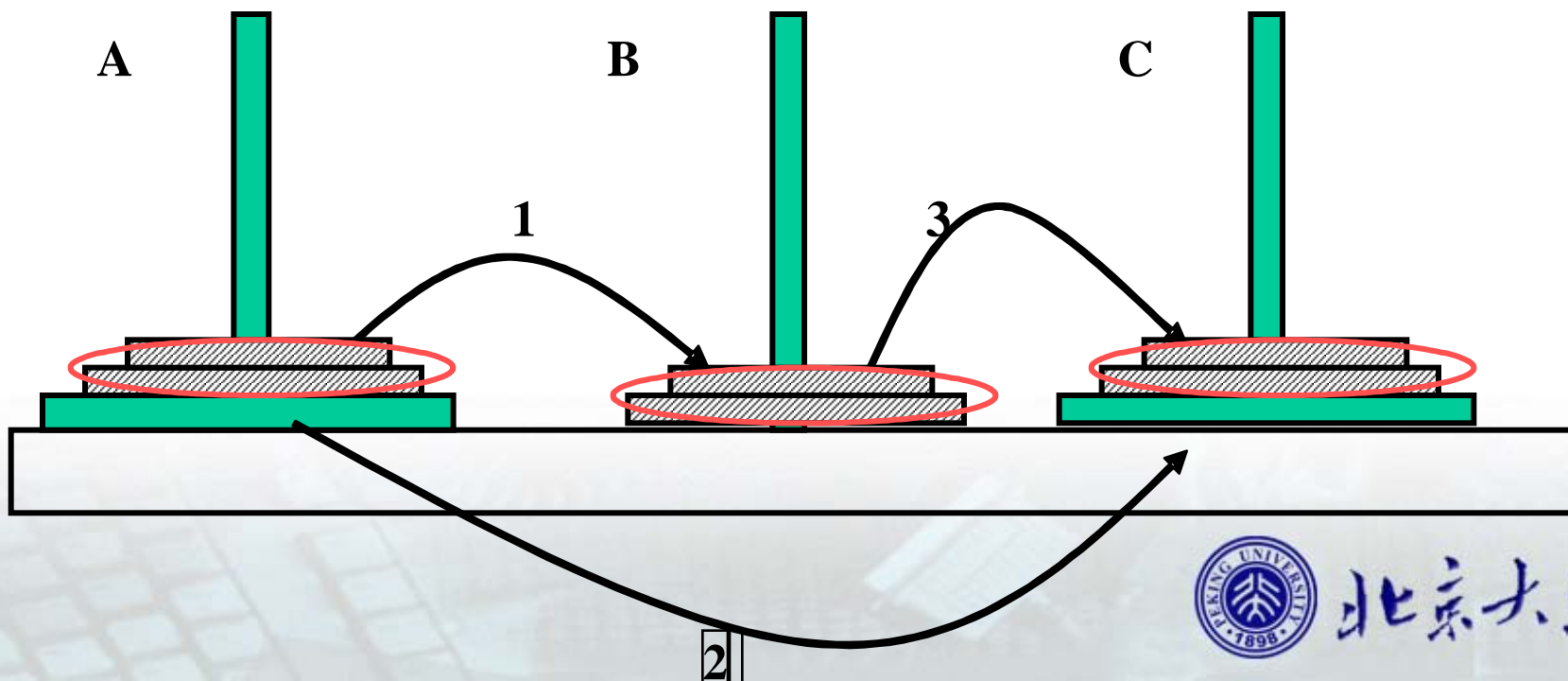
$\text{move}(2, A, C, B)$

第（2）步将3号盘从A移至C，一次到位。记为

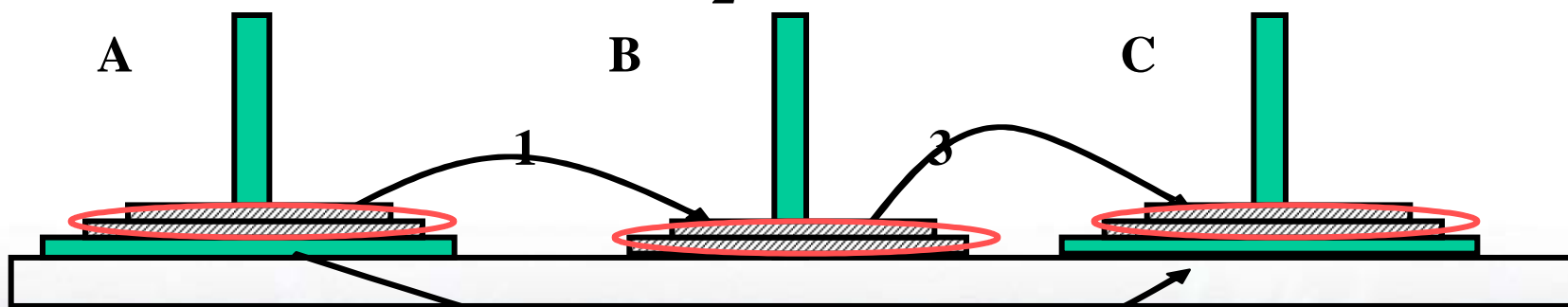
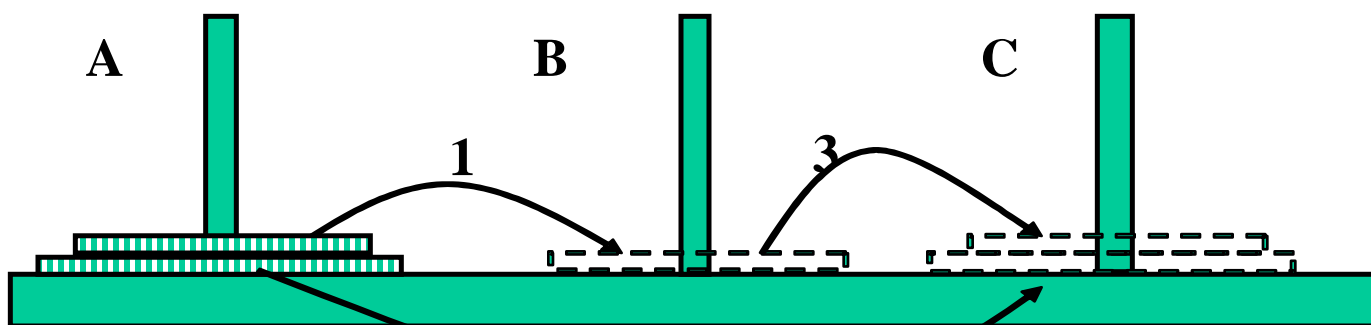
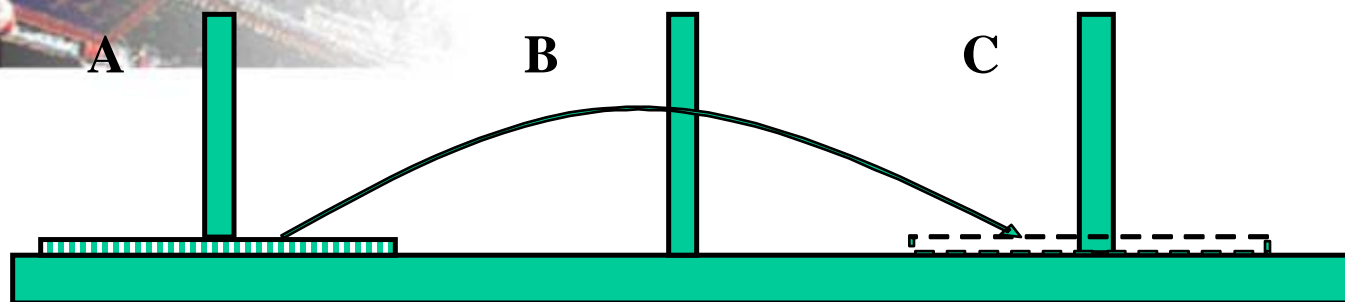
$\text{move } 3 \text{ from } A \text{ to } C$

第（3）步处于B上的作为一个整体的2只盘子，再移至C。这一步记为

$\text{move}(2, B, A, C)$



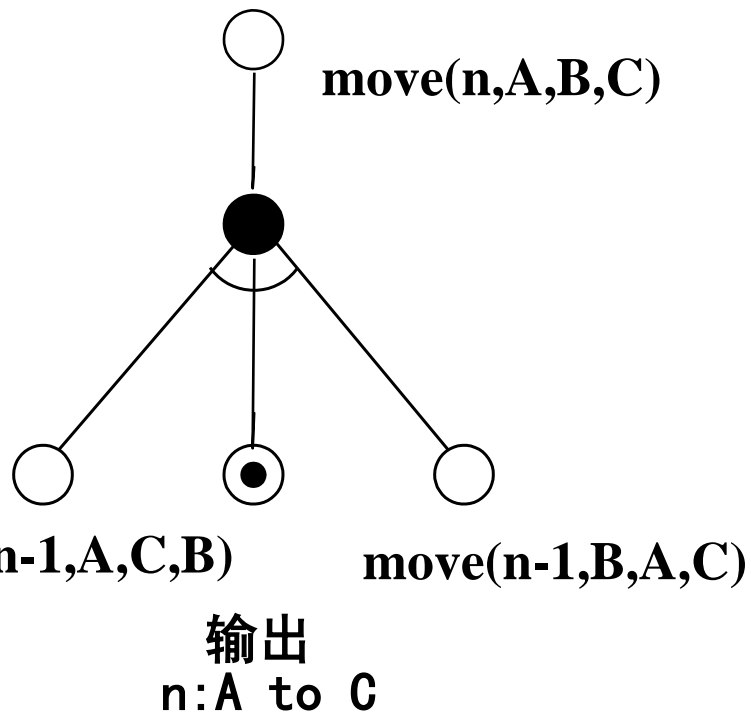
清华大学



递归经典问题——汉诺塔问题

■ $\text{move}(n, A, B, C)$ 分解为3步

- ◆ $\text{move}(n-1, A, C, B)$ 理解为将上面的 $n-1$ 只盘子作为一个整体从 A 经 C 移至 B;
- ◆ 输出 n : A to C, 理解将 n 号盘从 A 移至 C, 是直接可解结点;
- ◆ $\text{move}(n-1, B, A, C)$ 理解为将上面的 $n-1$ 只盘子作为一个整体从 B 经 A 移至 C。

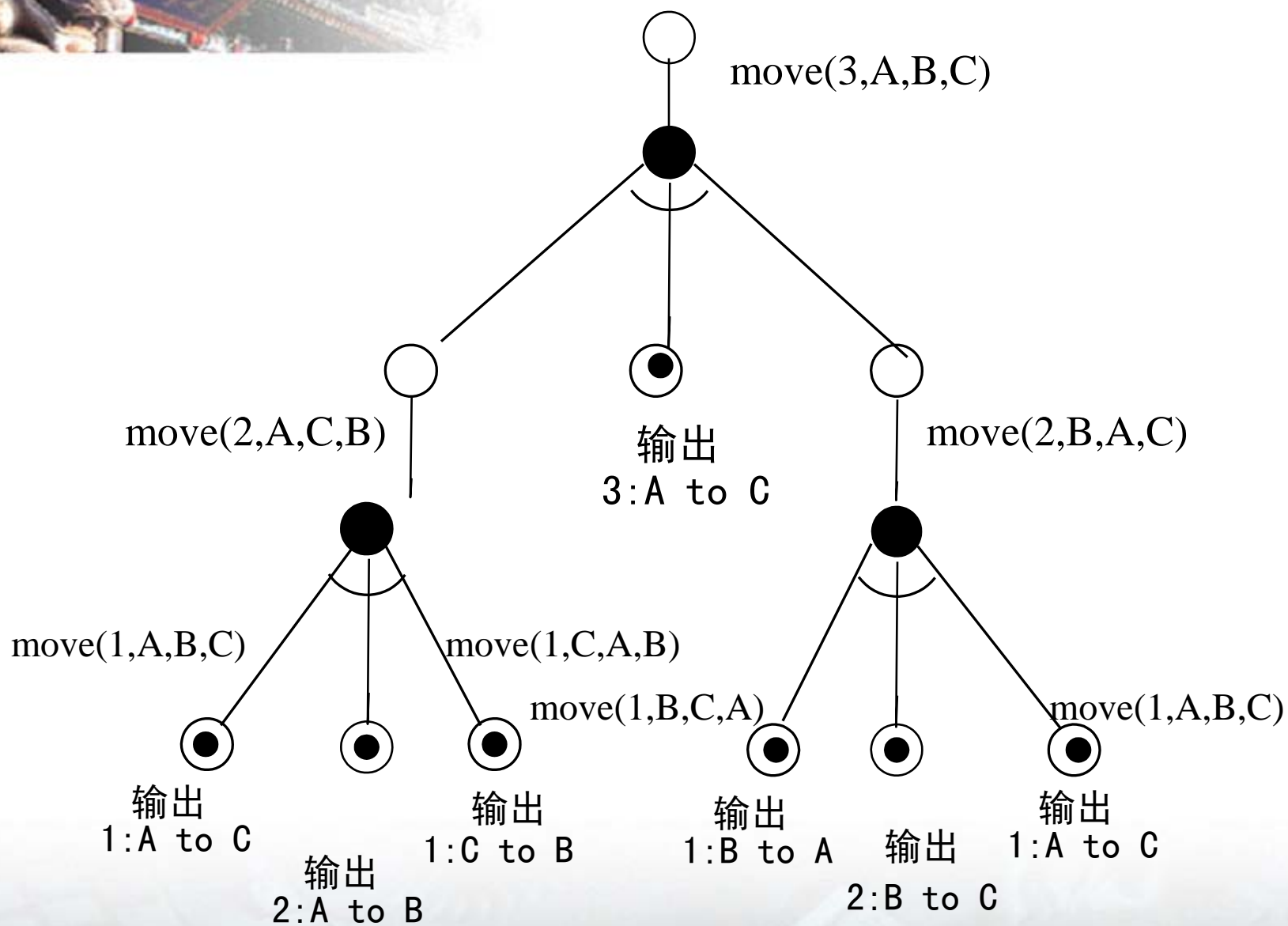


```

#include<iostream>
using namespace std;
void move(int m, char A, char B, char C) //表示将m个盘子从A经过B移动到C
{   if (m==1)                               //如果m为1,则为直接可解结点
    {
        cout<<"move 1# from"<<A<<" to "<<C<<endl; //直接可解结点
    }
    else                                     //如果不为1,则要调用move(m-1)
    {
        move(m-1,A,C,B);                  //递归调用move(m-1)
        cout<<"move 1# from"<<A<<" to "<<C<<endl; //直接可解结点
        move(m-1,B,A,C);                  //递归调用move(m-1)
    }
}

int main() {
    int n;                                //整型变量, n为盘数
    cout<<"请输入盘数n="<<endl;
    cin >> n;                             //输入盘子数目正整数n
    cout<<"在3根柱子上移"<<n<<"只盘的步骤为:"<<endl;
    move(n,'a','b','c');                  //调用函数 move(n,'a','b','c')
    return 0;
}

```



北京大学



好好想想,有没有问题?

谢谢!



清华大学