“奇技淫巧”话递归

王良 2018 年 9 月 26

“To Iterate is Human, to Recurs，Divine.” ---L. Peter Deutsch

“迭代是人，递归是神” 第一次见有人这样说，让我受伤的心得到些许安慰……

最近在琢磨算法，又见递归！ 这是个绕不过去的坎！ 当初，上大学时似懂非懂自欺欺人的蒙混过关，再次印证了那句名言：“出来混，迟早都是要还的……”。好吧，那就直面它！于是搜遍海内外，加上日思夜想，被这“奇技淫巧”折魔得真掉了不少头发（主要是 8 皇后问题~）。

大神王垠在谈程序语言最精华的原理时提到递归，并说递归比循环表达能力强很多，而且效率几乎一样！！！没有一定的内力，估计你很难理解他这句话，当然他说得没错！

务必要弄懂弄透它！抱着一股不服的拧劲和不死的初心，蒙生了收集所有经典递归案例写法作一汇总的想法。

如果您不能像大神一样一眼就看穿其本质并熟练（不自欺的）运用它，不妨一起来领略领略这“奇技淫巧”的各案之美， 由简到难，慢慢的你一定会觉得它越来越美！

如发现还有文中没收集到的经典例子，欢迎补充添加以泽后人，算积功德一件~

作者承诺：所有代码都经过测试，所有评论都是算法在脑中真真切切过了一遍之后的切肤反馈。

目录

经典递归例子汇总与点评： **2**
1. N！，求 N 的阶乘 **2**
2. 1+2+3+.....+n，求前 N 项和 **2**
3. Fibonacci 数列，F(n)=F(n-1)+F(n-2) **2**
4. GCD(a,b)，求最大公约数 **3**
5. Hanoi 塔，从 A 移到 C **3**
6. 回文数判定 **4**
7. 杨辉三角 **4**
8. 快速排序，二路归并 **5**
1. 快速排序 5
2. 二路归并 6
9. Bs Tree(二叉树的前，中，后序遍历） **6**
10. 全排列 **9**
11. 8 Queen problem **9**
12. 约瑟夫环问题 **11**
13. 求数组元素的最大值和最小值 **11**
14. 0/1 背包问题 **12**

经典递归例子汇总与点评：

1. N！，求 N 的阶乘

数学定义：

$$f(n)=\begin{cases}1, 当n=0\\ n*f(n-1), 当n>0时\end{cases}$$

```
//求N！
long int F( long int N)
{
        if(N==0)
                return 1;
        if(N>0)
                return N*F(N-1);
}

int main( int argv, char** argc){
        long int N;
        cin>>N;
        cout<<F(N)<<endl;
}
```

对着数学公式写代码是不是很容易？这个用递归比循环更易写，更容易理解！闭着眼睛，想想循环怎么写？...是不是有点啰嗦~？

2. 1+2+3+.....+n，求前 N 项和

这个是学编程用循环的入门级思维，用 for 语句太简单，但如果要以递归方式写，很多人可能又要卡一会儿了。若把它按数学公式的方式定义一下，和 N!一样，那递归就好写多了！
数学定义：

$$Sum(n)=\begin{cases}1, 当n=1\\ n+Sum(n-1), 当n>1时\end{cases}$$

```
int Sum(int N)
{

      if(N==1)
            return 1;
      if(N>1)
            return N+Sum(N-1);
}
```

按数学公式写代码，是不是让生活更美好！~

3. Fibonacci 数列，F(n)=F(n-1)+F(n-2)

数学定义：

$$\text{Fibonacci}(n)=\begin{cases}1, \text{当} n=1, \text{ 或} n=2\text{时} \\ \text{Fibonacci}(n-1)+\text{Fibonacci}(n-2), \text{当} n>2\text{时}\end{cases}$$

```
int Fibonacci(int N)
{
        if(N==1||N==2)
                return 1;
        if(N>2)
                return Fibonacci(N-1)+Fibonacci(N-2);
}
```

这个递归很好写，如果闭眼用循环写，估计要点时间，有几个变量需要耐心引入。

4. GCD(a,b)，求最大公约数

始祖殴基里德给出了辗转相除的递归原则，知道这个写起来就容易多了，但理解天才的想法是如何得来的还是要费点脑子的。

我曾试图去网上找找它的数学定义，很遗憾大都是些拗口的文字描述，代码反而容易理解一些。

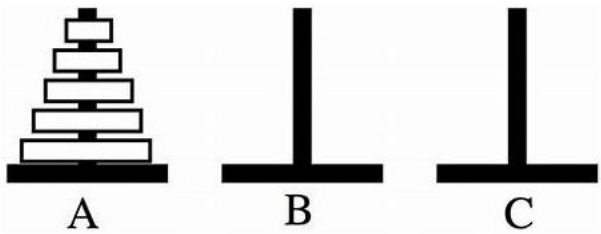
```
int GCD(int a, int b)
{
  if(a>0&&b>0){
          if(a%b==0){
                  return b;
          }else{
                  return GCD(b,a%b);
          }
    }
}
```

于是从代码中反推出其数学公式~:

$$\text{GCD}(a,b)=\begin{cases}b, \text{当} a \bmod b=0\text{时} \\ GCD(b, a \bmod b), \text{当} a \bmod b!=0\text{时}\end{cases}$$

5. Hanoi 塔，从 A 移到 C

真是佩服那个出题的和尚，高僧！！！当然会解题的也是高人。



如上图，要把 A 中所有圆盘经 B 辅助移到 C，移动过程中，要求园盘之上始终只能是比其小的圆盘。规则描述不复杂，但怎么把它转化成数学定义呢？懵！先从 code 入手？

```
 void Hanoi(int N, char source, char auxiliary, char target)
{

        if(N==1){
```

```
            cout<<"Move disk: "<<N<<" from "<<source<<" to "<<target<<endl;
        }else {
            Hanoi(N-1,source,target,auxiliary);
            cout<<"Move disk: "<<N<<" from "<<source<<" to "<<target<<endl;
            Hanoi(N-1,auxiliary,source,target);
        }
}

Hanoi(5,'A','B','C');
```

还是没法写成数学恒等式，为什么？因为在其递归终点不再是返回值以供调用者使用，而是执行了一次操作，依次返回过程中都是再执行一次操作，不是数量表达关系，因此较难用数学定义语言描述这类问题。

6. 回文数判定

不用递归，够你死一丢脑细胞的。回文数是指前后对称的数，如（1，121，12321 等）。假设数字都以字符串的形式存储，为了大数判断和一般的通用回文判断，采用这种形式

```
bool isPali(string S, int startindex)
{
        if(S.size()==1||startindex>=S.size()-1-startindex){
            return true;
        }
        if(S[startindex]!=S[S.size()-1-startindex]){
            return false;
        }

        return isPali(S, startindex+1);
}

int main(){
        string S;
        cin>>S;
        cout<<isPali(S,0)<<endl;
}
```

试试把它数学定义式写出来：

$$\text{isPali}(S,n)=\begin{cases}1,\ \text{当}S.size\text{是1时}\\ 1,\ \text{当}n\geq S.size/2\text{时}\\ 0,\ \text{当}S[n]!=S[S.size-n-1]\text{时}\\ isPali(S,n+1),\text{其它}\end{cases}$$

7. 杨辉三角

老祖宗蛮厉害，但这样说，似乎有点类我党自夸之嫌~

```
                    1
                  1   1
                1   2   1
              1   3   3   1
            1   4   6   4   1
          1   5   10  10  5   1
        1   6   15  20  15  6   1
      1   7   21  35  35  21  7   1
    1   8   28  56  70  56  28  8   1
1   9   36  84  126 126 84  36  9   1
```

把它转化成(row,col)直角形式如下

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
```

要得到i行，j列 数字的值，递归算法如下：

```
int GetVOfYangHui(int row, int col)
{
	if(col<=row && col>=0){
		if(col==0||row==col){
			return 1;
		}else{
			return GetVOfYangHui(row-1,col-1)+GetVOfYangHui(row-1,col);
		}
	}else {
		return 0;
	}
}
```

试试把它数学定义式写出来：

$$\text{GetVOfYangHui}(i,j)=\begin{cases}1,\ \text{当}j=0\text{且}i>=0\text{时}\\ 1,\ \text{当}i=j\text{且}i>=0\text{时}\\ \text{GetVOfYangHui}(i-1,j-1)+\text{GetVOfYangHui}(i-1,j),\ \text{当}1<i<j\text{时}\\ 0,\ \text{其它}\end{cases}$$

8. 快速排序，二路归并

让你闷头写一个，是不是也有点难度~?

1. *快速排序*

主框架用递归的思维很简单，稍麻烦一点的是分割 DPart，需要用到一点编程的技巧。

```
int DPart(int* A, int start, int end)
{
	int key=A[end];
	int index=end;
	while(start<end){

		while(A[start]<key){ start++;}
		while(A[end]>=key){end--;}
		if(start<end){ swap(A[start],A[end]);}
		else{
			break;
		}

	}
	swap(A[start],A[index]);
	return start;
}

void QuickSort(int* A, int start, int end)
{
	if(start>end||start<0||end<0){
		return;
	} else {
		int index=DPart(A,start,end);
		QuickSort(A,start,index-1);
		QuickSort(A,index+1,end);
```

```
        }
}
```

2. *二路归并*

主框架用递归的思维也很简单，关键在写 Merge 时，需要用到一点点编程的技巧。

```
void Merge(int A[], int low, int mid, int high)
{
        int n1= mid-low+1;
        int n2= high-mid;

        int L[n1],R[n2];
        for(int i=0;i<n1;i++)
                L[i]=A[i+low];

        for(int j=0;j<n2;j++)
                R[j]=A[j+mid+1];

        int i=0,j=0,k=low;

        while(i!=n1 && j!= n2)
        {
           if(L[i] <= R[j])
                A[k++] = L[i++];
           else
                A[k++] = R[j++];
        }

          while(i < n1)
             A[k++] = L[i++];
          while(j < n2)
            A[k++] = R[j++];

}

void MergeSort(int A[], int low, int high)
{
        if(low<high){
                int mid = (low+high)/2;
                MergeSort(A,low,mid);
                MergeSort(A,mid+1,high);

                Merge(A,low,mid,high);
        }
}
```

9. Bs Tree(二叉树的前，中，后序遍历）

这个可算是搞数据结构设计的人把递归思想发挥到极致的经典案例吧？

这部分 code 有些冗长，为了完整性，还是把它全贴出来。主要目的是体会其前，中，后序遍历的递归写法。为了建树方便，我们以数据输入顺序按层序方式建树，它需要用到队列技巧（与递归无关，暂不讨论），同时加入一个层序遍历方法来验证输入。注意**层序遍历**很难用递归方法实现，我思考了很久都没有结果，如果你有想到，一定告知一声，万谢！
（mathmad@163.com)

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

using namespace std;

struct Node {
        int data;
        Node* left;
        Node* right;
};

class Btree
```

```
{
    public:
        Btree();
        ~Btree();

        Node *createNode(int data);
        Node *GetRoot();
        void insert(Node *newNode);
        void destroy_tree();
        void levOrder(Node *root);
        void preOrder(Node *root);
        void midOrder(Node *root);
        void posOrder(Node *root);
    private:
        void destroy_tree(Node *leaf);
        Node *root;
        queue<Node *> q;
};

Btree::Btree()
{
        root=NULL;
}
Btree::~Btree()
{
        if(root!=NULL){
                destroy_tree(root);
        }
}

void Btree::destroy_tree(Node *leaf)
{
        if(leaf!=NULL)
        {
                destroy_tree(leaf->left);
                destroy_tree(leaf->right);
                delete leaf;
        }
}
Node* Btree::createNode(int data)
{
        Node* n=new Node;
        n->left=NULL;
        n->right=NULL;
        n->data=data;
        return n;
}
Node* Btree::GetRoot()
{
        return root;
}

void Btree::insert(Node *newnode)
{
        if(NULL==root){
                root=newnode;
                q.push(root);
        }else{
                Node *f=q.front();
                if(f->left&&f->right){q.pop(); f=q.front();}
                if(!f->left){ f->left=newnode;}
                else if(!f->right){ f->right=newnode;}
                q.push(newnode);

        }
}

void Btree::levOrder(Node* root)
{
        if(root){
                queue<Node*> Q;
                Q.push(root);

                while(!Q.empty()){
                        Node *d=Q.front();
                        cout<<d->data<<" ";
                        Q.pop();
                        if(d->left){Q.push(d->left);}
                        if(d->right){Q.push(d->right);}
                }
```

```
        }
}

void Btree::preOrder(Node* root)
{
        if(NULL==root) { return;}
        else{
                cout<<root->data<<" ";
                preOrder(root->left);
                preOrder(root->right);
        }
}

void Btree::midOrder(Node *root)
{
        if(NULL==root) { return;}
        else{
                midOrder(root->left);
                cout<<root->data<<" ";
                midOrder(root->right);
        }
}

void Btree::posOrder(Node *root)
{
        if(NULL==root) { return;}
        else{
                posOrder(root->left);
                posOrder(root->right);
                cout<<root->data<<" ";
        }
}

int main( int argv, char** argc){

        Btree btree;
        Node * newnode;
        int data;
        cout<<"Please input a sequences number to create a Complete Binary Tree:"<<endl;

        do{
                cin>>data;
                newnode=btree.createNode(data);
                btree.insert(newnode);
        }while(getchar()!='\n');

        cout<<"BTree Level order is:"<<endl;
        btree.levOrder(btree.GetRoot());
        cout<<endl<<"Pre Order is:"<<endl;
        btree.preOrder(btree.GetRoot());
        cout<<endl<<"mid Order is:"<<endl;
        btree.midOrder(btree.GetRoot());
        cout<<endl<<"pos Order is:"<<endl;
        btree.posOrder(btree.GetRoot());
        cout<<endl;

}
```

用 g++ 编译上述代码(g++ BTree.cpp)，测试输出如下：

```
[root@wl Geeks]# ./a.out
Please input a sequences number to create a Complete Binary Tree:
1 2 3 4 5 6 7 8
BTree Level order is:
1 2 3 4 5 6 7 8
Pre Order is:
1 2 4 8 5 3 6 7
mid Order is:
8 4 2 5 1 6 3 7
pos Order is:
8 4 5 2 6 7 3 1
```

10. 全排列

这个有点难想清楚！特别是第二个 swap 交换！

N 个元素的全排列，有高中数学基础的人都易知道它总共有 N!(阶乘）种。若要全部打印出来，当 N>4 时还是有一定麻烦，特别是当用循环思路正面强攻时，会让人陷入无尽的深渊！

下面以 5 个数字为例简述其原理：
假设数据集为{1, 2, 3, 4, 5}， 如何编写全排列的递归算法？

1、首先看最后两个数 4, 5。 它们的全排列为 4 5 和 5 4，即以 4 开头的 5 的全排列和以 5 开头的 4 的全排列。
由于一个数的全排列就是其本身，从而得到以上结果。
2、再看后三个数 3, 4, 5。它们的全排列为 3 4 5、3 5 4、4 3 5、4 5 3、5 4 3、5 3 4 六组数。是以 3 开头的和 4,5 的全排列的组合、以 4 开头的和 3,5 的全排列的组合和以 5 开头的和 4,3 的全排列的组合，即 k 与(k+1,..N)的全排列组合加上 k’与（k+1,...N)的全排列组合，其中 k’是 k 与(k+1,...N)的任一置换。

网上通用一般描述为，设一组数 p = {r1, r2, r3, ... ,rn}，全排列为 perm(p)，令 pn = p - {rn}。因此 perm(p) = r1perm(p1), r2perm(p2), r3perm(p3), ... , rnperm(pn)。当 n = 1 时 perm(p} = r1。为了更容易理解，将整组数中的所有的数分别与第一个数交换，这样就总是在处理后 n-1 个数的全排列。（下面给出 C++版，以字符串为数据集的全排列算法）

```
#include <iostream>
using namespace std;
/*
	全排列算法改进
	Author: Liang 2018-09-21

*/

int Perm(string s, int k, int m) {
	static int n=0;

	if (s!= ""&& m>=k && m<=s.size()-1 && k>=0){
		if(k == m){
			cout<<s<<endl;
			n++;
		}else {
			Perm(s, k+1, m);

			for(int i = k+1; i <= m; i++) {
				swap(s[k],s[i]);
				Perm(s, k+1, m);
				swap(s[k],s[i]);
			}
		}
	}
	return n;
}

int main(){
	string str;
	cin>>str;
	int n=str.size();
	cout<<"total: "<<Perm(str,0,n-1)<<endl;
}
```

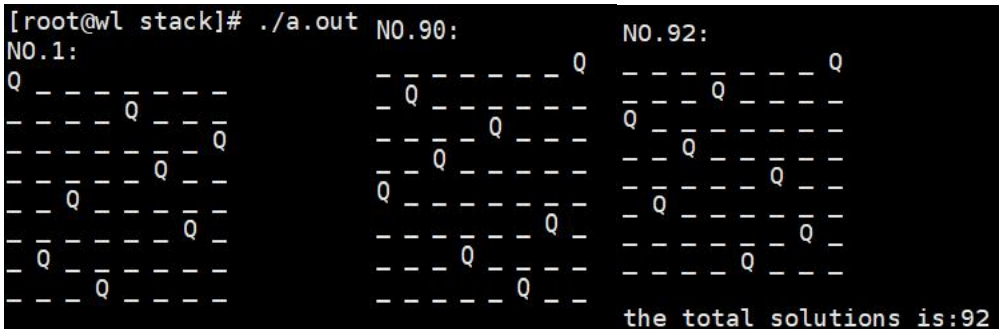
11. 8 Queen problem

国际象棋棋盘上放 8 个皇后，问有多少种放法（皇后的走法是水平垂直线和对角线位置可以互相攻击，皇后就是这么牛~，比象棋的车还厉害） 著名的 8 皇后问题，这个 Gauss 都只得出 76 种解（实际有 92 种），一想到这，我心理就会平衡一点~

8 皇后问题后来演变成 N 皇后问题，其中 N=1 时有一个解， 2，3 时无解，N=4 时有两个解，N=5 时比 N=6 的解还多。这说明什么问题？ 一个“老婆”最稳定，2 个 3 个后宫没法处理，挺到 4 个时就会有办法，5 个“老婆”比 6 个老婆还好处置！超过 6 个后，就是多多益善了，要想处理 8 个皇后，高斯都没想清楚！可见后宫是多么难治啊，哈哈，闲扯远了~

```
#include <iostream>
#include <cstdlib>
using namespace std;
#define StackSize 8 // could be set to 1, 4, 5, 6, 7,....N

int ans=0;
int top=-1;
int ColOfRow[StackSize]; //index represent the row, value is col

void Push(int col)
{
    top++;
    ColOfRow[top]=col;
}

void Pop()
{
    top--;
}
// check put a Queen to position(row, col) is safe or not.
bool isPosSafe(int row, int col)
{
    for(int i=0;i<row;i++)
    {
      //check col is ok, row is increase invoke, same is impossible
        if(col==ColOfRow[i]){
            return false;
        }
      // Y=kX+b, k=1 or k=-1, the Queen will attack each other
      if(abs(col-ColOfRow[i])==(row-i)){
            return false;
      }
    }
    return true;
}

void PrintBoard()
{
    cout<<"NO."<<ans<<":"<<endl;
    for(int i=0;i<StackSize;i++)
    {
        for(int j=0;j<ColOfRow[i];j++)
            cout<<"_ ";
        cout<<"Q";
        for(int j=StackSize-1;j>ColOfRow[i];j--)
            cout<<" _";
        cout<<endl;
    }
    cout<<endl;
}
// Should be start from 0 row, since the start point will impact future steps desision
void PlaceQueen(int row)
{
    for (int col=0;col<StackSize;col++)
    {
        Push(col);
        if (isPosSafe(row,col))
        {
            if (row<StackSize-1)
                PlaceQueen(row+1);
            else
            {
                ans++;
                PrintBoard();
            }
        }
        Pop();
```

```
    }
}

int main()
{
    // Since Recursion invoke, start point should be 0, the first row
    PlaceQueen(0);
    cout<<"the total solutions is:"<<ans<<endl;
    return 0;
}
```



12. 约瑟夫环问题

编号为 1,2,3....N 的 N 个人围成一圈，从第一个人开始，从 1 报数，数到 K 的出列（有些版本把其描为自杀），下一个人重新从 1 开始报，到 K 再出列，持续下去问最后幸存者编号（估且认为他们是自杀游戏），如果 N 是变化的或者很大的数，应该怎么处理？

这个网上有很多分析，总之，如果没有数学家搞出这个递推公式来，循环怕是要让人想破头！有人说用链表实现很简单，但若 N 是一个很大很大的数，链表就不好玩了，空间成本难以为计！

有了数学递归公式，程序写起来就太简单了，甚至都不用过脑子~，让这个递归过程在脑中运行一遍试试，还是有点伤脑筋的。

数学定义：

$$f(n,k) = ((f(n-1,k) + k - 1) \bmod n) + 1, \text{ with } f(1,k) = 1\,,$$

```
#include<iostream>

using namespace std;

int Jose(long int n, int k)
{
    if(n==1){
        return 1;
    }else {
        return (Jose(n-1,k)+k-1)%n + 1;
    }
}

int main(){
    int N, K;
    cin>>N>>K;
    cout<<Jose(i,K)<<endl;
}
```

13. 求数组元素的最大值和最小值

好了，如果你翻山越岭看完了前面所有，并有一定的递归思想和理解，碰到这个问题就很简单了，再也不用用冒泡两两对比的思维方式了。

```
int FindMax(int* A, int N)
{

	if(N==1){
		return A[0];
	}else {

		return (FindMax(A,N-1)>A[N-1])?FindMax(A,N-1):A[N-1];
	}
}

int FindMin(int* A, int N)
{

	if(N==1){
		return A[0];
	}else {

		return (FindMin(A,N-1)<A[N-1])?FindMin(A,N-1):A[N-1];
	}
}


int main( int argv, char** argc){
	int N;
	cin>>N;
	int A[N];

	for(int i=0;i<N;i++)
		cin>>A[i];

	cout<<"Max: "<<FindMax(A,N)<<endl;
	cout<<"Min: "<<FindMin(A,N)<<endl;

	return 0;
}
```

14. 0/1 背包问题

0/1 背包是动态规划的一个经典例子，但是 DP 理解起来有一定困难。不用 DP，常规思维一般会想到穷举，如果不用递归，循环写起来会很麻烦很麻烦（几乎不可能）。N 个物品就有 2 的 N 次方种组合（每一种商品，选或者不选），下面给出递归的写法（当然动态规划效率是更优的方案）

*问题描述：*有 N 件商品，每一件商品都有对应的重量 Weight[i]和价格 Value[i], 给您一个承重上限是 BagCapacity 的背包，问最多能装下多少价值的货？

```
#include<iostream>
using namespace std;

int KnapSack(int BagCapacity, int Weight[], int Value[], int N)
{
   // Base Case
   if (N == 0 || BagCapacity == 0)
       return 0;

   // If weight of the Nth item is more thaN Knapsack capacity W, theN
   // this item caNNot be included in the optimal solutioN
   if (Weight[N-1] > BagCapacity)
       return KnapSack(BagCapacity, Weight, Value, N-1);

   // Return the maximum of two cases:
   // (1) Nth item included
   // (2) Not included
   else return max( Value[N-1] + KnapSack(BagCapacity-Weight[N-1], Weight, Value, N-1),
```

```
                    KnapSack(BagCapacity, Weight, Value, N-1)
                );
}

int main()
{
    int N, BagCapacity;
    cout<<"please input N kinds of items, Number first, follow item's weight, then each value"<<endl;
      cin>>N;
    int Weight[N];
    int Value[N];
    for(int i=0;i<N;i++)
      cin>>Weight[i];
    for(int i=0;i<N;i++)
      cin>>Value[i];
    cout<<"please input the bag's capacity"<<endl;
      cin>>BagCapacity;
    cout<<KnapSack(BagCapacity, Weight, Value, N)<<endl;
    return 0;
}
```