递归 2

递归思想是把一个大型复杂的问题层层转化为一个与原问题相似的规模较小的问题来求解。递归策略只需少量的程序就可描述出解题过程所需要的多次重复计算，大大地减少了程序的代码量。

掌握递归思维，需要善于将大问题拆分成由其次小问题求解的问题，从而建立递推式，进行递归或递推求解。例如：求 1+2+3+……+n 这个问题，传统数学思维是直接利用等差数列求解，即（1+n）*n/2，然而对于递归思维，将该问题描述为 f(n)，要求 f(n)，如果我们知道 f(n-1)，即 1+2+3+……+(n-1)的和，那么在 f(n-1)的基础上加上 n 即可求得 f(n)，所以 f(n)=f(n-1)+n。而 f(n-1)则进一步递归求解。

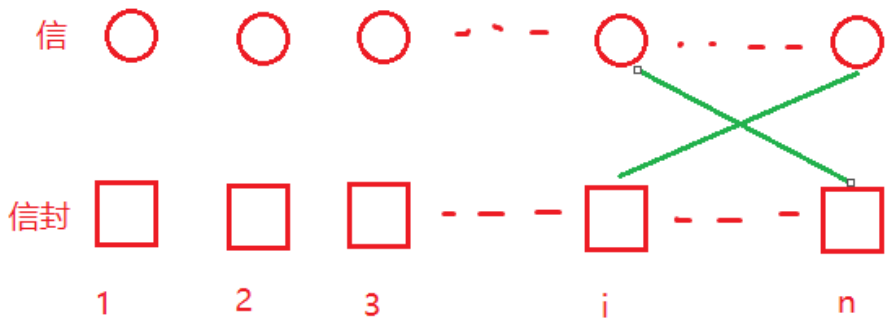
使用递归思维，我们可以将重点放在推导递推式上，而不需考虑计算问题，因为计算的问题交给程序解决。

例：装信问题

某人写了 N 封信和 N 个信封，结果所有的信都装错了信封。求所有的信都装错信封共有多少种不同情况。

对于此题，我们模拟求解，如果有 1 封信，则方案数为 0，如果有 2 封信，则方案数为 1，如果有 3 封信，则方案数为 2。依次模拟，我们能快速找到规律吗？其实很难。但是利用递归思维，我们则能很容易解决此题。

我们设求解的问题为 f(n)，表示 n 封信错装的方案数。考虑最后一封信，即第 n 封信，它不能装在第 n 个信封，假如它装在前面 n-1 个信封中的任意一个信封 i（1<=i<n），那么考虑第 i 封信，如果第 i 封信装到第 n 个信封，则 i 和 n 构成错装，剩下 n-2 封信进行错装，即 f(n-2);



如果第 i 封信不装在第 n 个信封，那么问题和第 i 封信不装在第 i 个信封性质类似，所以问题变成有 n-1 封信进行错装，即 f(n-1);

综上分析，当 n 选择了 i 进行错装，方案数有 f(n-1)+f(n-2)，但是 n 有 n-1 个信封可以选择，所以 f(n)=(n-1)*(f(n-1)+f(n-2))。有了这个递推式，我们则可以很轻松求解出问题的解。

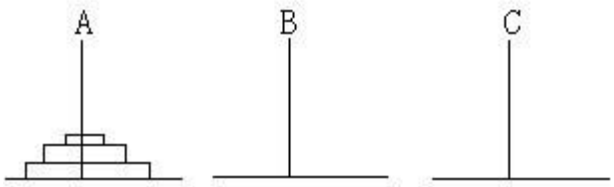
```
#include<iostream>
using namespace std;
long long n,f[20];
int main(){
    cin>>n;
    f[1]=0;
    f[2]=1;
    for(int i=3;i<=n;i++){
        f[i]=(f[i-1]+f[i-2])*(i-1);
```

```
    }
    cout<<f[n];
}
```

以上问题的分析，正是递归思维的特点，求解一个问题，将其转化成子问题，推导出原问题和子问题的关系，构建出递推式，然后进一步递归求解子问题。

移梵塔

【题目描述】
有三根柱 A，B，C 在柱 A 上有 N 块盘片，所有盘片都是大的在下面，小片能放在大片上面。现要将 A 上的 N 块片移到 C 柱上，每次只能移动一片，而且在同一根柱子上必须保持上面的盘片比下面的盘片小，请输出移动的步骤。



【输入】
输入整数 n，表示有 n 块盘片
【输出】
输出移动的步骤
【样例输入】
3
【样例输出】
1 A->C
2 A->B
3 C->B
4 A->C
5 B->A
6 B->C
7 A->C
【分析】
本题需要用到递归思维。将 n 个盘片移走的次数设为 f(n)，那么将前 n-1 个盘片移走，需要 f(n-1)次，然后将第 n 个盘片移走，需要 1 次，再将移走的 n-1 个盘片已过去，需要 f(n-1)，所以总的次数 f(n)=f(n-1)+1+f(n-1)，即 f(n)=2*f(n-1)+1。如果单纯求次数，利用这个递归函数即可求，但是题目要求输出步骤，则需要在递归过程中将步骤输出。
【代码】

```
#include<iostream>
using namespace std;
int num,n;
void move(int n,char s,char m,char t){//函数表示将 n 个盘从 s 柱借助 m 柱移到 t 柱
    if(n==0)return;
```

```
    move(n-1,s,t,m); //先将 n-1 个盘从 s 柱借助 t 柱移到 m 柱
    cout<<++num<<" "<<s<<"->"<<t<<endl;//直接将第 n 个盘从 s 柱移到 t 柱
    move(n-1,m,s,t); //将移走的 n-1 个盘从 m 柱借助 s 柱移到 t 柱
}
int main(){
    cin>>n;
    move(n,'A','B','C');
}
```

汉诺 4 塔

【题目描述】
“汉诺塔”，是一个众所周知的古老游戏。现在我们把问题稍微改变一下：如果一共有 4 根柱子， 而不是 3 根，那么至少需要移动盘子多少次，才能把所有的盘子从第 1 根柱子移动到第 4 根柱子上呢？
【输入】
一个正整数 n。（0<n<=60）
【输出】
一个正整数，表示把 n 个盘子从第 1 根柱子移动到第 4 根柱子需要的最少移动次数。
【样例输入】
3
【样例输出】
5
【分析】
使用递归思维分析此题。我们设 4 根柱子移动 n 个盘子的方案数为 f(n)，三个柱子移动 n 个盘子的方案数为 h(n)。考虑一次移动，我们可以将 A 柱上 n 个盘子中的 i 个盘子移动到 B 或 C 柱（方便起见，我们设为 C 柱），移动的方案数为 f(i)，那么 i 柱上剩下 n-i 个盘子，需要从 A 柱借助 B 柱移动到 D 柱，则方案数为 h（n-i），最后将 C 柱上的 i 个盘子再移动到 D 柱，方案数为 f(i)。那么，移动 n 个盘子的方案数为 f(n)=2*f(i)+h(n-i)，对吗？i 是多少？i 的取值范围为 0<=i<n，i 为 0 时，即用不移动，问题转化为三柱问题，i 为 n-1 时，问题变为剩下最后一个盘子一步从 A 移到 D 柱。那么对于 i 的各种取值，怎么确定最少的移动次数呢？在各种取值中找最少的，即 f(n)=min(2*f(i)+h(n-i))|0<=i<n。
边界：f(0)=0。
【代码】

```
#include<iostream>
#include<cstring>
using namespace std;
long long n,f[60],h[60];
int solve(int n){
    if(n==0)return 0;
    if(f[n]!=f[0])return f[n];
    for(int i=0;i<n;i++){
        f[n]=min(f[n],solve(i)*2+h[n-i]);
```

```
    }
    return f[n];
}
int main(){
    memset(f,127,sizeof(f));
    cin>>n;
    h[1]=1;
    for(int i=2;i<=n;i++)h[i]=2*h[i-1]+1;  //预先求出三塔的步数
    cout<<solve(n);
}
```

数的计算

【题目描述】
我们要求找出具有下列性质数的个数(包含输入的自然数 n):
先输入一个自然数 n(n<=1000),然后对此自然数按照如下方法进行处理:
1.不作任何处理;
2.在它的左边加上一个自然数,但该自然数不能超过原数的一半;
3.加上数后,继续按此规则进行处理,直到不能再加自然数为止.
【输入】
自然数 n
【输出】
满足条件的数的个数
【样例输入】
6
【样例输出】
6
【提示】
满足条件的数为 6、16、26、126、36、136
【分析】
模拟过程，对于数 n，可以接在它左边的数有 n/2 个，对这 n/2 个数继续递归下去求即可，如下：

```
#include<iostream>
using namespace std;
int n,ans;
int dfs(int n){
    int sum=0;    //存储方案数
    for(int i=1;i<=n/2;i++)sum+=dfs(i);    //模拟前一位的方案，且递归处理下去
    return ++sum;    //方案数加 1，表示它自身也是一种方案
}
int main(){
    cin>>n;
    cout<<dfs(n);
```

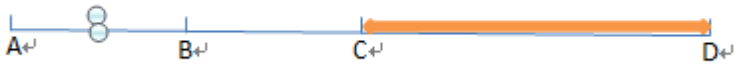
```
12. }
```

对于 n=1000，这个代码会超时，需要记忆化优化。
【代码】

```
1.  #include<iostream>
2.  using  namespace  std;
3.  int  n,ans,f[1001];
4.  int  dfs(int  n){
5.          if(f[n])return  f[n];
6.          for(int  i=1;i<=n/2;i++)f[n]+=dfs(i);
7.          return  ++f[n];
8.  }
9.  int  main(){
10.         cin>>n;
11.         cout<<dfs(n);
12. }
```

程序龙的密码

【题目描述】
程序龙有 oiClass 的管理权限，同学们就一直想寻找机会骗取密码，以便黑掉 oiClass。
有一天，程序龙在课堂上和同学们打赌输了，于是他不得不告诉同学们 oiClass 的管理密码。但是程序龙很机（jiao）智（hua），他并不会直接告诉同学们密码，而是将密码的生成方法说出来。
他说，密码的生成方法是这样的，设集合 A 中 A={1,2,...,n}，B 为 A 子集。对于 B 中任意一个元素 x，2x 均不在集合 B 中。B 中元素数目最大值即为密码。
说完，程序龙 45 度角仰望天空，嘴角漏出一丝轻笑。
但他忘了，你是学过编程的高手，你能轻松编程算出密码，对吧？
【输入】
一行，一个整数 n(1<=n<=maxlongint)
【输出】
只有一个整数 m，表示 B 中元素最大值
【样例输入】
100
【样例输出】
67
【分析】
我们可以以样例来说明，样例为 100，那么从 51 到 100 的数 i，肯定不存在 2*i，因为 51*2=101，超过 100 了，所以这里至少有 50 个数满足条件。我们再考虑 1 到 50 的区间，这个区间的数 i，都存在 2*i<=100，那么这些数都不能用了吗？不是，在 1 到 50 的区间内，如果我们删除 26 到 50 的数，那么 1 到 25 之间数 i 就不一定都存在 2*i 了，那么我们就可以递归处理 1 到 25 区间有多少个满足要求的数。
题目也可以理解为：



集合 A 即为数轴 AD,现在研究集合 B，首先对折数轴得到 CD 满足条件，依次向下研究 AC ，去掉 BC，可以得到 AB，再向下研究 AB，依次下去可以得到解。
需要特别注意 n 为奇数的情况。
【代码】

```
#include<iostream>
using namespace std;
int n,ans;
int dfs(int n){
        if(n==0)return 0;
        return (n+1)/2+dfs(n/4);
}
int main(){
        cin>>n;
        cout<<dfs(n);
}
```

装箱问题

【题目描述】
有一个箱子容量为 v(正整数，o≤v≤20000)，同时有 n 个物品(o≤n≤30)，每个物品有一个体积（正整数)。要求从 n 个物品中，任取若干个装入箱内，使箱子的剩余空间为最小。
【输入】
第一行，一个整数，表示箱子容量;
第二行，一个整数，表示有 n 个物品;
接下来 n 行，分别表示这 n 个物品的各自体积。
【输出】
一个整数，表示箱子剩余空间。
【样例输入】
24
6
8
3
12
7
9
7
【样例输出】
0
【分析】

要使箱子的剩余空间最小，其实等价于 v-箱子的最大装载空间。我们设 f(v,n)表示对于 v 的空间有 n 个物品的最大装载空间。考虑第 n 个物品，如果将第 n 个物品装进箱子，则挤掉 a[n]的空间，也即是说，对于能装载的空间，剩下 v-a[n]的空间给前面的物品装载，好处是获得 a[n]的装载空间，即问题变为 f(v-a[n],n-1)+a[n]，f(v-a[n],n-1)表示前面 n-1 个物品有 v-a[n]的空间进行装载所能获得的最大装载空间；如果第 n 个物品不装进箱子，那么问题就变为 f(v,n-1)，即前面 n-1 个物品有 v 的空间进行装载所能获得的最大装载空间。对于第 n 个物品，它有两种决策，装和不装，在这两种决策中选择最大的装载空间，问题即变为 f(v,n)=max(f(v-a[n],n-1)+a[n],f(v,n-1))。

边界：f(0,n)=0，即没有剩余空间了，

f(v,0)=0，即没有物品可装了。

细节：对于每个物品，不一定都有两种决策，装和不装，如果当前物品的体积为 a[n]，而剩余空间 v<a[n]，则显然，第 n 件物品不能装，因为空间不够。

【代码】

```
#include<iostream>
using namespace std;
int v,n,a[31];
int dfs(int n,int v){
    if(n==0||v==0)return 0;
    if(a[n]>v)return dfs(n-1,v);  //不能装
    else return max(dfs(n-1,v),dfs(n-1,v-a[n])+a[n]);
}
int main(){
    cin>>v>>n;
    for(int i=1;i<=n;i++)cin>>a[i];
    cout<<v-dfs(n,v);
}
```

2 的幂次方

【题目描述】

任何一个正整数都可以用 2 的幂次方表示.

例如:137=2^7+2^3+2^0

同时约定次方用括号来表示,即 a^b 可表示为 a(b)

由此可知,137 可表示为:2(7)+2(3)+2(0)

进一步:7=2^2+2+2^0 (2^1 用 2 表示)

3=2+2^0

所以最后 137 可表示为:2(2(2)+2+2(0))+2(2+2(0))+2(0)

又如:1315=2^10+2^8+2^5+2+1

所以 1315 最后可表示为:2(2(2+2(0))+2)+2(2(2+2(0)))+2(2(2)+2(0))+2+2(0)

【输入】

正整数(n<=10^15)

【输出】

符合约定的 n 的 0,2 表示(在表示中不能有空格)

【输入样例 1】
137
【输入样例 2】
1315
【输出样例 1】
2(2(2)+2+2(0))+2(2+2(0))+2(0)
【输出样例 2】
2(2(2+2(0))+2)+2(2(2+2(0)))+2(2(2)+2(0))+2+2(0)
【分析】
递归，可以通过位运算计算出 n 的幂次方，对幂次方递归处理。注意输出的细节处理。
【代码】

```
#include<iostream>
#include<cstring>
using namespace std;
long long n;
void work(long long n){
    if(n==0){   //边界处理
        cout<<0;
        return;
    }
    bool a[50];
    memset(a,false,sizeof(a));
    int len=0;
    while(n!=0){  //二进制分离 n 的幂
        a[len++]=n&1;
        n>>=1;
    }
    for(int i=len-1;i>=0;i--){  //从高位开始枚举
        if(a[i]==false)continue;  //该位为 false 表示没有该幂次方
        if(i!=len-1)cout<<"+";  //特判第一项输出
        cout<<"2";
        if(i!=1){  //特判指数为 1 的情况
            cout<<"(";
            work(i);
            cout<<")";
        }
    }
}
int main(){
    cin>>n;
    work(n);
}
```