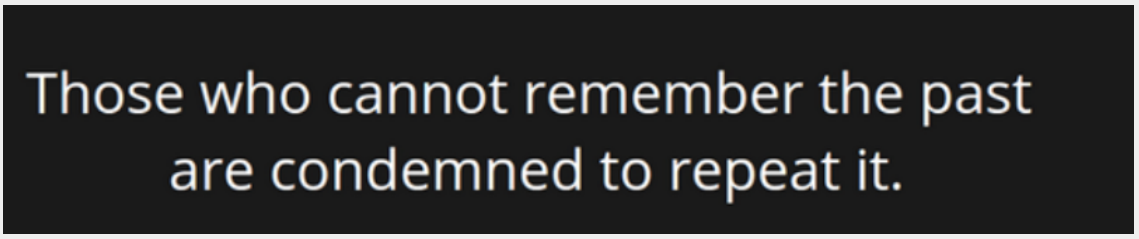
一、动态规划




```
A * "1+1+1+1+1+1+1+1 =? " *

A : "上面等式的值是多少"
B : *计算* "8!"

A *在上面等式的左边写上 "1+" *
A : "此时等式的值为多少"
B : *quickly* "9!"
A : "你怎么这么快就知道答案了"
A : "只要在8的基础上加1就行了"
A : "所以你不用重新计算因为你记住了第一个等式的值为8!动态规划算法也可以说是 '记住求过的解来节省时间'"
```

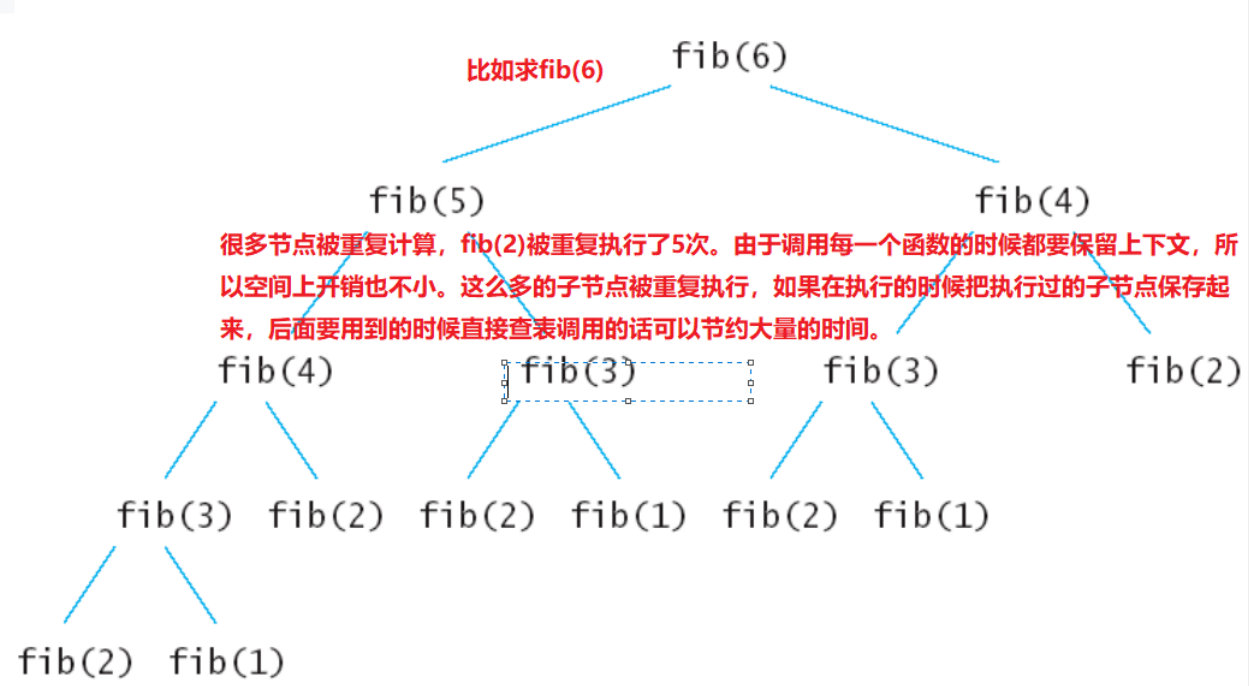
1.1 问题提出

```
Fibonacci (n) = 1;    n = 0
Fibonacci (n) = 1;    n = 1
Fibonacci (n) = Fibonacci(n-1) + Fibonacci(n-2)
```

斐波那契数列

```
public int fib(int n)
{
    if(n<=0)
        return 0;
    if(n==1)
        return 1;
    return fib( n-1)+fib(n-2);
}
```

递归 算法求解



1.2 自顶向下的备忘录法

```java
//自顶向下备忘录法
public class FibonacciSolution {
    public static int Fibonacci(int n)
    {
            if(n<=0)
                return n;
            int []Memo=new int[n+1];//1.memo：备忘录，n+1项，存储从0到第n项的斐波那契值
            for(int i=0;i<=n;i++)
                Memo[i]=-1;//初始化为-1
            return fib(n, Memo);
    }
    public static int fib(int n,int []Memo)
    {

        if(Memo[n]!=-1)
          return Memo[n];
        //2.如果已经求出了fib（n）的值直接返回，否则将求出的值保存在Memo备忘录中。
        if(n<=2)
          Memo[n]=1;
        //3.递归，不过记住了每次计算过程中的每一项的斐波那契数列的值
        else Memo[n]=fib( n-1,Memo)+fib(n-2,Memo);

        return Memo[n];
    }

    @Test
    public void test1(){
        System.out.println(Fibonacci(46));//int类型最多能计算到第46项
        System.out.println(Integer.MAX_VALUE);
    }

}
```

上面还是用到了递归。

1.3 自底向上的动态规划

```java
public static int fib(int n)
{
    //其实核心就是备忘录，记录了每一项的值
        if(n<=0)
            return n;
        //用数组记录每一项的值，回归最淳朴的数列定义而已
        //而不是去用数列递归
        //当然，其实不必要存储每一项的值，因为没必要
        //所以可以很自然的优化成空间复杂度为1
        int []Memo=new int[n+1];
        Memo[0]=0;
        Memo[1]=1;
        for(int i=2;i<=n;i++)
        {
            Memo[i]=Memo[i-1]+Memo[i-2];
        }
        return Memo[n];
}
```

函数递归

不是说这个方法，是下面的这个方法fib2

```java
public static int fib2(int n)
{
    //其实核心就是备忘录，记录了每一项的值
        if(n<=0)
            return n;
        int pre2=0,pre1=1;
        int fib=0;
        for(int i=2;i<=n;i++)
        {
            fib=pre1+pre2;
            pre2=pre1;
            pre1=fib;
        }
        return fib;
}
```

优化了一下空间复杂度