1.绪论

迭代与递归

尾递归

邓俊辉

deng@tsinghua.edu.cn

Tail Recursion

❖ 递归算法易于理解和实现，但空间（甚至时间）效率低

在讲求效率时，应将递归改写为等价的迭代形式

❖ `fac(n) { return (1 > n) ? 1 : n * fac(n-1); }`

❖
```
fac(n) {
    if (1 > n) return 1;
    else return n * fac(n - 1); //tail recursion
}
```

❖ 尾递归：最后一步是递归调用

最简单的递归模式，可便捷地改写

Tail Recursion

❖

```
fac(n) {  /* 递归 */

   if (1 > n) return 1;

   else return n*fac(n-1);

} //O(n)时间 + O(n)空间
```

```
fac(n) {  /* 统一转换为迭代 */

   int f = 1; //记录子问题的解

next: //转向标志，模拟递归调用

   if (1 > n) return f;

   f *= n--;

   goto next; //模拟递归返回

} //O(n)时间 + O(1)空间
```

```
fac(n) {  /* 简捷 */

   int f = 1;

   while (1 < n)

      f *= n--;

   return f;

} //O(n)时间 + O(1)空间
```

课后

- 做递归跟踪分析时，为什么递归调用语句本身可不统计？
- 试用递归跟踪法，分析fib()二分递归版的复杂度

  通过递归跟踪，解释该版本复杂度过高的原因
- 递归算法的空间复杂度，主要取决于什么因素？
- 本节数组求和问题的两个（线性和二分）递归算法

  时间复杂度相同，空间呢？