A2 参考题解

很多同学导出的PDF里面一堆数学公式全部都是LaTeX公式，根本看不了……建议之后导出的时候check一下PDF

1 Array-based Data Structures

1.1 Implement two Stacks in one Array

这道题的思路很简单，只需要把两个stack的栈底设置为数组的两端，然后两者都往数组中间增长. 大部分同学都能做对（尽管有个别同学把栈底设在数组的中间……），此处不赘述细节. 这里附上一些同学的解答供大家参考.

```
class doubleStackArray:
    A[1:n] # an array that stores the
    top1 := 0 # the top of Stack1 which increases when PUSH and decreases when POP
    top2 := n+1 # the top of Stack2 which decreases when PUSH and increases when
PUSH
    isFull():
        return top1>top2
    push1(val):
        if isFull():
            return STATE_OVERFLOW
        else:
            top1 += 1
            A[top1] = val
    push2(val):
        if isFull():
            return STATE_OVERFLOW
        else:
            top2 -= 1
            A[top2] = val
    pop1():
        if top1==0:
            return STATE_UNDERFLOW
        else:
            top1 -=1
    pop2():
        if top2==n+1:
            return STATE_OVERFLOW
        else:
            top2 += 1
```

图1. 某位大佬的解答

p.s. 关于伪代码：我们不要求大家用伪代码完成题目，用C++/Python等也可以，只要能把意思传达到位即可. 不过我看到有些使用CLRS系列伪代码的同学可能会犯一些小错误，例如数组位置从0开始（应该从1开始）.

2 Linked List

2.1 Reverse a Linked List

反转链表本身是一道很常见的编程题，我们只需要从前往后不断让当前节点的next指向它的前一个节点即可. 这个时候，因为改变next会导致找不到下一个节点，所以应当用一个指针变量记录原来的next；前一个节点同理，也需要进行记录. 这道题大家同样没什么问题，这里提供一位同学的解答供参考:

```
Reserve()
    prev, cur:=NULL, head
    while cur
        temp=cur.next
        cur.next=prev
        prev=cur
        cur=temp
    return pre
```

图2. 某位大佬的解答

3 Divide-and-Conquer & Recursion

3.1 Multiply Matrices with Squaring Algorithm

这道题可以有不同的思路解, 比方说我第一时间想到的就是这个式子:

$$(A+B)^2 = A^2 + 2AB + B^2$$

只需要计算$(A+B)^2$, A^2和B^2, 就可以得到AB了. 剩下的矩阵加法和减法都是$O(n)$级别的操作, 故可以保证总开销为$\Theta(n^\alpha)$.

不少同学用了其它思路, 例如构造矩阵$C \in \mathbb{R}^{2n} \times \mathbb{R}^{2n}$:

$$C = \begin{bmatrix} \mathbf{0} & A \\ B & \mathbf{0} \end{bmatrix}$$

此时计算C^2可以得到:

$$C^2 = \begin{bmatrix} \mathbf{0} & AB \\ BA & \mathbf{0} \end{bmatrix}$$

该算法避免了矩阵加减法, 要计算平方的矩阵大了一倍, 但是这都不影响其渐进复杂度为$\Theta(n^\alpha)$.

3.2 Sort the Pancakes

这道题如果只是设计一个符合题目要求的算法是不难的, 但是如果要设计一个最优的算法会很难, 我们这里仅给出最trivial的算法. 其中, 对于最坏情况的分析可能不够tight, 因为我没有给出一个很严谨的理论分析.

1. 设计一个$O(n)$ flips的算法并不难, 这里可以采用选择排序的思路: 每一次我用常数时间把最大的那张pancake翻到最下面, 然后问题规模减一, 然后不断重复直至问题规模为0即可. 而想要把一张最大的pancake从中间某个地方翻到最下面最多需要两步: 第一步, 把它翻到最上面; 第二步, 将全体翻转, 它就会跑到最下面; 伪代码可以这么实现:

```
def pancakeSort(S, n):
  """
  S[1:n]: The stack of pancakes.
          S[1] is the top of the stack
  """
  if n == 1: # base case
    return

  m = findLargestPancake(S, n)
  if m != n:
    if m != 1
```

```
    flip(S, 1, m) # Flip pancakes from S[1] to S[m]
  # Now S[1] is the largest pancake
  flip(S, 1, n)
  # Now S[n] is the largest pancake
pancakeSort(S, n-1)
```

不难计算出该方法的worst case是$2n-2$次flips (base case不需要进行flip).

2. 如果要让每张pancake的burned side向下, 我们只需要在每次把最大那张pancake翻到最上面的时候, check一下它是否是burned side向上；如果不是, 那我们单独把它翻转一下, 然后再把它翻到最底下. 不难计算出该方法的worst case是$3n-2$次flips (最后只需要检查最上面的pancake是否burned side朝下, 如果不是那就再翻一次).

   这道题似乎有同学理解有误, 这个地方不是使得所有pancake的burned side朝下即可, 还需要满足第二题, 也就是大小有序的条件.

3.3 Calculate the Inversions

如果没有这个$O(n\log n)$的限制, 数逆序对倒是有一个挺方便的方法, 就是直接使用冒泡排序, 交换多少次就是多少个逆序对. 受到这个方法的启发, 我们可以观察到:

> 当我们在Merge两个数组时, 比方说数组A和数组B；当我们把数组B中一个元素取出来的时候, 数组A中剩余的元素, 就是比数组B该元素大的所有元素.

根据reduction的思路, 我们可以把计算一个数组A的逆序对数量分解成:

1. 把数组A分成A_1和A_2两半（跟归并排序的分法是一样的）.
2. 分别计算A_1和A_2各自的逆序对数量.
3. 再计算横跨A_1和A_2的逆序对, 即$\#(A[i], A[j])$使得$A[i]\in A_1 \land A[j]\in A_2 \land A[i] > A[j]$.
   这一步的计算利用到我们上面提到的观察结果, 也就是计算逆序对数时, 我们只需要在每次选取A_2元素时, 获取一下A_1中剩余元素即可；用伪代码表示一下大概如下:

```
def merge(A, a, b, c, Temp):
  """
  A1 = A[a:b]
  A2 = A[b+1:c]
  """
  i = a
  j = b+1
  k = 0
  count = 0
  while i <= b and j <= c:
    if A[i] > A[j]:
      Temp[k++] = A[j]
      # We modify here
      count += b - i + 1 # calculate the number of inversions
    else:
      ... # same as Mergesort
  ... # same as Mergesort
  return count
```

4. 把第2步和第3步的数量加起来即可.

而第2步可以进行递归调用, 因此只需要把递归基确定下来即可.