

远古计算机题解

清华大学 吴凯路

闲话

► 普通选手WC体验：

- T1这什么神仙题？？怎么还两棵树都是随机的，我不做了
- T3这是什么神仙交互？没做过交互题，不会，丢了
- T2，诶，第一个点就有20分，看起来美滋滋，我来看看

子任务编号	1	2	3	4	5
集训队分值	10	15	15	30	30
非集训队分值	20	20	20	20	20

然后呢？

- ▶ 随机阅读了超级冗长的题面之后，选手有了许多问题

向标准输出输出两个数会覆盖前一个数字吗？

为什么这个checker运行出现段错误？
为什么我运行下来错了？

这个是不是我输出正确了就停了？ 为什么我运行下来不对，
什么都没有输出？

这个graph.in是什么啊？

这个result.out为什么是空白的？

一个周期是什么？

这checker真不友好

这是一个源代码长达15K的checker
(测评用的长达25K)
(出题人觉得就该让选手写个解释器，占80%的分)

不过detail.out还是非常友好的，debug速度++



你已经是个大checker了
要学会自己debug了

所以这题该怎么玩呢？

► 子任务1:

- 这个子任务是读完题，正确使用难用的checker后即可得分
- 选手：啊哈！白送20分！看我两分钟拿20分
- 20min later
- 选手：这个checker怎么还在不对



子任务1

- ▶ 只需要输入如下几十个字符，即可获得20分！点击即得！白送不要钱！
 - ▶ node 1
 - ▶ read 0 a
 - ▶ write a 0

子任务2

- ▶ 选手：不就一个斐波那契数列嘛，递推谁不会啊！
- ▶ 10min later
- ▶ $F_i = F_{i-1} + F_{i-2}$ ，似乎我需要两个变量存F，还需要一个循环变量。。。怎么只有两个寄存器啊！

子任务2

- ▶ 0%:

- ▶ 我们把代码展开
- ▶ if $n = 0$ then print 1
- ▶ if $n = 1$ then print 1
- ▶ if $n = 2$ then print 2
- ▶ ...

子任务2

- ▶ 30%:
 - ▶ 我会二分!
 - ▶ if `a == 10` then ...
 - ▶ else if `a < 10` then `jmpto ...`
 - ▶ else `jmpto ...`

子任务2

- ▶ 60%:
- ▶ 题面`jmp reg`的意思是不是可以`jmp a`啊？！
- ▶ `read 0 a`
- ▶ `add a a`
- ▶ `add a 4`
- ▶ `jmp a`
- ▶ 然后就可以实现立即寻址，跑到第 $a*2+4$ 条指令开始执行
- ▶ `write xxx 0`
- ▶ `jmp 1`

子任务2

- ▶ 100%:
 - ▶ 诶，题面“以最早输出正确答案的周期为准”
 - ▶ 于是
 - ▶ read 0 a
 - ▶ add a 3
 - ▶ jmp a
 - ▶ 每个分支只占1行
 - ▶ write xxx 0

子任务3

- ▶ 30%:
 - ▶ 一次一个数传遍所有节点
- ▶ 60%:
 - ▶ 最优解上出了偏差
- ▶ 100%:
 - ▶ 找到一条最短路，然后走最短路传递数据

子任务4

- ▶ 第 1 到 50 号节点分别输入 1 个数，将这 50 个数从 51 到 100 号节点输出，输出顺序任意，每个节点输出数字个数任意。
- ▶ 网络流？
- ▶ No
- ▶ 考察选手面对任意问题的解决能力（暴露题答本质）

子任务4

▶ 30%:

- ▶ 找到最短路，全部从最短路传输

▶ 60%:

- ▶ 朴素贪心
- ▶ 贪心给每个点匹配一个终点，直接限定一个节点只能用于传输一个数字

▶ 100%:

- ▶ 优秀的贪心
- ▶ 如何充分利用节点呢?
- ▶ 分层图，然后使用费用流类似每次寻找一个最短路的贪心方法，可以在7个周期内完成。

子任务5

▶ 30%:

- ▶ 修改一下上一子任务的满分程序，可在31个周期内运行完成

▶ 60%:

- ▶ 输出一下每个点度数，发现64个点度数都是4，以及度数为2的点只有4个，其余度数为3
- ▶ $8*8=64$
- ▶ 诶，这不是个网格图？！
- ▶ 发现这是个重新编号的网格图，1~10号点在第一排，91~100号点在最后一排。1号在左上角，100在右下角。
- ▶ 开始构造答案！
- ▶ 一个普通的构造用时22个周期

子任务5

- ▶ 100%:
 - ▶ 一个优秀的构造，用时21个周期
 - ▶ 接下来为一个6*6的样例

1	2	3	4	5	6

1					
	2	3	4	5	6

	1				
	2				
		3	4	5	6

		1			
		2			
		3			
			4		
				5	6

			1		
			2		
			3		
			4		
				5	
					6

				1
				2
			3	
			4	
			5	
				6

					1
				2	
			3		
			4		
			5		
			6		

				2	1
			3		
		4			
		5			
		6			

			3	2	1
		4			
	5				
	6				

		4	3	2	1
	5				
6					

	5	4	3	2	1
6					

6	5	4	3	2	1