Analysis of task nod2

Statement, analysis, solutions, idea: Dimitar Shapatov; Solutions: Kiril Zashev

Subtask 1

Trivially we will use one of the registers for storing our value.

Subtask 2

Of course, the solution is a bruteforce $O(NQ)$, but how are we going to implement it?

Here we already have to delve deeper into the particularities of this language. First we need a loop with Q iterations for the queries. We can do this by using one register for Q, decreasing it by 1 on every iteration and, when it reaches 0, doing a `JZ` to a `RETURN` instruction. On every iteration we read t, l, r and check whether $t = 1$ (we can put it in RES, subtract 1 and do a `JZ`); if so we jump to our loop for queries of type 1, otherwise we continue to the one for type 2. The two loops are implemented in a similar way – we process index l of the array, increase l, compare it with r, if it is greater we continue to the next query and, if necessary, we print the answer.

Subtask 3

From here on there are two approaches, one of them using respectively one or two Fenwick trees, but we will look at the other approach, since it is easier to understand (and significantly harder to implement). In this subtask the constraint hints at an ordinary segment tree.

There are many particularities in the implementation, but the main one is that we have no simple way of doing recursive functions. We have to implement the recursion ourselves, by maintaining a stack of the parameters with which our function has been called – for a vertex, its left and right bound and one more parameter which we will look at below. In the `update` function, the value that we add can be stored at a chosen index in the array for convenience, since it is the same for every execution of the function within a given query. Besides that we also have to maintain a pointer to the current index in our stack. When we enter the function, we add our parameters to the stack, and when we return to the parent, i.e. we do a `return` from the function, we remove ours and call the function with the parameters of the parent. In fact, in computers recursion is implemented in the same way – a call stack.

But another problem arises – since we work only with `JZ`, `JNZ`, there is no way to call our function twice within itself, because we do not know how to get back, since we call it

from the beginning. Therefore as a fourth parameter to the function we have a number between 0 and 2 inclusive.

- A value of 0 means that we have entered this function and have not fully processed any child. If we see a value of 0, we start processing the left child.
- A value of 1 means that we have processed 1 child – the left one. If we see 1, we process our right child.
- A value of 2 means that our vertex is ready. If we see 2, we have to return to our parent, remove ourselves from the stack, and this way we say that our subtree is ready.

An important detail is that when we return to the parent, we increase its fourth parameter before calling it with it.

For convenience, it is not necessary to keep any values returned by some call of the `query` function; instead we can simply keep in some register the sum of the vertices which are fully covered by our query.

Subtask 4

It only remains to implement lazy propagation, which we can do by allocating some other part of the array for lazy values. The function which pushes the lazy value of a vertex to its children can be implemented directly inside the other functions answering the queries. It can also be implemented as a separate function, and since it is not recursive, it is much easier to implement than the other two functions.

At the end of this function an unconditional jump (`MOV RES 0, JZ x`) has to be made to the function which called it. Since `x` is different for each of the three places where the function is called, it is most convenient to make three separate functions, but this problem can also be fixed with a few checks inside the function itself, which determine at which line it will return after its execution.

Comment

If it was not obvious, the main goal of the task is to waste the contestants' time. Solving this task is not particularly worth it, unless your name is Dimitar Shapatov and you are forced to write it 🙂