Анализ на задача nod2

Условие, анализ, решения, идея: Димитър Шапатов; Решения: Кирил Зашев

Подзадача 1

Тривиално ще използваме единия регистър за запазване на стойността ни.

Подзадача 2

Разбира се, решението е bruteforce $O(NQ)$, но как ще го имплементираме?

Тук вече се налага да вникнем повече в особеностите на този език. Първо ни трябва цикъл с Q итерации за заявките. Това можем да направим, като използваме един регистър за Q, на всяка итерация го намаляме с 1 и когато стигне 0 правим `JZ` към инструкция `RETURN`. На всяка итерация въвеждаме t, l, r и проверяваме дали $t = 1$ (можем да го сложим в RES, да извадим 1 и да направим `JZ`), ако е така скачаме към цикъла ни за заявки тип 1, иначе продължаваме към този за тип 2. Двата цикъла се имплементират по подобен начин – обработваме индекс l от масива, увеличаваме l, сравняваме го с r, ако е по-голям продължаваме към следващата заявка и, ако се налага, извеждаме отговора.

Подзадача 3

От тук нататък има два подхода, като един от тях използва съответно едно или две дървета на Fenwick, но ще разгледаме другия подход, тъй като е по-лесен за разбиране (и значително по-труден за имплементация). В тази подзадача ограничението ни подсказва за обикновено сегментно дърво.

Има много особености в имплементацията, но основната е, че нямаме прост начин за рекурсивни функции. Трябва да имплементираме рекурсия сами, като поддържаме стек от параметри, с които функцията ни е била извикана – за връх, негова лява и дясна граница и още един параметър, който ще разгледаме по-долу. При функцията `update`, самата стойност, която добавяме, може да се запази в избран индекс в масива за удобство, тъй като тя е еднаква за всяко изпълнение на функцията в дадена заявка. Освен това трябва да поддържаме и показалка към текущия индекс в стека ни. Когато влезем в функцията, добавяме нашите параметри в стека, а когато се връщаме към родителя, т.е. правим `return` от функцията, махаме нашите и викаме функцията с параметрите на родителя. Всъщност, в компютрите рекурсията се имплементира по същия начин – call stack.

Но възниква друг проблем – тъй като работим само с `JZ`, `JNZ`, няма как да извикаме

два пъти функцията си в себе си, защото ние не знаем как да се върнем, понеже я викаме отначало. Затова като четвърти параметър към функцията имаме число между 0 и 2 включително.

- Стойност 0 означава, че сме влезнали в тази функция и не сме обработили нито едно дете напълно. Ако видим стойност 0, започваме да обработваме лявото дете.
- Стойност 1 означава, че сме обработили 1 дете – лявото. Ако видим 1, обработваме дясното ни дете.
- Стойност 2 означава, че върхът ни е готов. Ако видим 2, трябва да се върнем към родителя ни, да се махнем от стека и така казваме, че поддървото ни е готово.

Важен детайл е, че когато се връщаме към родителя, увеличаваме неговия четвърти параметър, преди да го викнем с него.

За удобство, не е нужно да си пазим някакви стойности, върнати от някое извикване на функцията `query`, а можем просто в някой регистър да пазим сумата на върховете, които са изцяло покрити от заявката ни.

Подзадача 4

Остана само да имплементираме lazy propagation, което можем да направим, като си заделим някоя друга част от масива за lazy стойности. Функцията, която бута lazy стойността на връх към децата му, може да се имплементира директно в другите функции, отговарящи заявките. Също може и да се имплементира като отделна функция и тъй като тя не е рекурсивна, е много по-лесна за имплементация, отколкото другите две функции.

В края на тази функция трябва да се направи unconditional jump (`MOV RES 0, JZ x`) към функцията, която я е извикала. Тъй като `x` е различно за всяко от трите места, където функцията е извикана, най-удобно е да се направят три отделни функции, но този проблем също може да се оправи с няколко проверки в самата функция, които определят на кой ред тя ще се върне след изпълнението си.

Коментар

Ако не беше очевидно, основната цел на задачата е да губи времето на състезателите. Решаването на тази задача не си заслужава особено, освен ако се казваш Димитър Шапатов и си принуден да я напишеш 🙂