В тази задача можем да представим мрежата като пълен ориентиран граф. За да е хъб един връх трябва всички негови ребра да са изходящи.

Ограниченията на задачата не позволяват да проверяваме всички ребра на всеки връх или да обръщаме всяко ребро на обърнат връх.

В тази задача е удобно да представим графа чрез матрица на съседство. За да оптимизираме обръщането на ребро можем да използваме булев масив reversed[N], за който reversed[i] = 1 ако ребрата на връх i са обърнати и 0 в противен случай. Така имаме насочено ребро (i, j) ⬄ G[i][j] ^ reversed[i] ^ reversed[j] == 1, където G е матрицата на съседство на графа, а ^ е операцията XOR (изключващо или).

Важното наблюдение, което трябва да направим е, че един връх няма как да е хъб, ако има входящо ребро. В такъв случай можем с всяка проверка на ребро да изключваме по един кандидат за хъб. Т.е. ако проверяваме връх i и имаме ребро (i, j), то няма нужда да проверяваме j. Ако пък нямаме ребро (i, j), то имаме ребро (j, i), т.е. няма нужда да проверяваме останалите ребра на i – можем направо да „скочим“ на j. В крайна сметка ще останем само с един връх, на който трябва да проверим всички ребра, за да сме сигурни, че е хъб. Т.е. с O(n) операции намираме кандидат за хъб и с още O(n) го проверяваме – общата сложност на проверката остава O(n). За най-лесно писане можем да започнем търсенето си, като връх 0 ни е кандидатът за хъб и последователно проверяваме връзките му с връх 1, 2, 3, … като променяме кандидатът за хъб при входящо ребро (Вж. авторско решение).