Let us define m variables which denotes the number of times each button was toggled.
Variables:
$C = C_1, C_2...C_j, ..., C_m$, where C_j is a non-negative integer.

Light i stays lit if it was toggled odd number of times. Hence, for each of the i light bulbs, we have the following constraint (total n constraints):

Factors: $f_1, f_2, ...f_i, ..., f_n$ with each $f_i = 1$

where each factor/constraint is:

$$f_i(C) = [(\sum_{k}^{m} C_k * \mathbb{1}[i \epsilon T_k])\%2 = 1]$$

Though we sum over all buttons, the indicator function is essentially ensuring that the summation is only performed over a subset of buttons where for each of these buttons k, $i \epsilon T_k$. Hence, we can also write the same as:

$$f_i(C) = [(\sum_{k;\text{ such that } i \epsilon T_k}^{m} C_k)\%2 = 1]$$

Let this subset by denoted by S_i. Then S_i is the scope of f_i and $|S_i|$ is its arity.

Also, note that the real count of how many times a button was toggled does not matter. What matters is whether the button was toggled odd or even number of times which can be used as an additional optional info to aid solving the CST. In that case, the variables will just be binary valued (0 or 1).

i) Checking all possible assignments:

X_1	X_2	X_3	$t_1 = (X_1 \oplus X_2)$	$t2 = (X_2 \oplus X_3)$	$W = t1 * t2$
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	1	**1**
0	1	1	1	0	0
1	0	0	1	0	0
1	0	1	1	1	**1**
1	1	0	0	1	0
1	1	1	0	0	0

The product of the factors is 1 for 2 assignments. Hence, there are two consistent assignments mentioned above.
1. $X_1 = 0, X_2 = 1, X_3 = 0$
2. $X_1 = 1, X_2 = 0, X_3 = 1$

ii) Call stack:

- backtrack({}, 1, {x1:{0, 1}, x2:{0, 1}, x3: {0, 1}})
  - backtrack({x1:0}, 1, {x2:{0, 1}, x3: {0, 1}})
    - backtrack({x1:0, x3:0}, 1, {x2:{0, 1}})
      - backtrack({x1:0, x3:0, x2:1}, 1, {})
    - backtrack({x1:0, x3:1}, 1, {x2:{0, 1}})
  - backtrack({x1:1}, 1, {x2:{0, 1}, x3: {0, 1}})
    - backtrack({x1:1, x3:0}, 1, {x2:{0, 1}})
    - backtrack({x1:1, x3:1}, 1, {x2:{0, 1}})
      - backtrack({x1:1, x3:1, x2:0}, 1, {})

Total number of calls = 9

iii) Call stack:

- backtrack({}, 1, {x1:{0, 1}, x2:{0, 1}, x3: {0, 1}})
  - backtrack({x1:0}, 1, {x2:{1}, x3: {0}})
    - backtrack({x1:0, x3:0}, 1, {x2:{1}})
      - backtrack({x1:0, x3:0, x2:1}, 1, {})
  - backtrack({x1:1}, 1, {x2:{0}, x3: {1}})
    - backtrack({x1:1, x3:1}, 1, {x2:{0}})
      - backtrack({x1:1, x3:1, x2:0}, 1, {})

Total number of calls = 7.

Let us take few auxiliary variables S_1, S_2 and S_3 such that S_i is supposed to hold the pair denoting the sum till $i-1$ elements and i elements.

Hence, $S_i[1] = S_i[0] + X_i$
Also, $S_i[0] = S_{i-1}[1]$

Also, $S_1[0] = 0$ since it is the sum of 0 elements by definition.

To summarize,
Auxiliary Variables: S_1, S_2, S_3

Unary Factors:
- $S_1[0] = 0$
- $S_3[1] \leq K$

Binary Factors:
- $S_2[0] = S_1[1]$
- $S_3[0] = S_2[1]$
- $S_1[1] = S_1[0] + X_1$
- $S_2[1] = S_2[0] + X_2$
- $S_3[1] = S_3[0] + X_3$

It is easy to verify this works since $S_i[1]$ is fetching the sum of first i elements and we restrict $S_3[1]$ to be $\leq K$. Further, we copy $S_{i-1}[1]$ to $S_i[0]$ so that all our factors are binary factors so as to prevent creation of 3-ary factor (between previous sum, current term and new sum).

Profile
minUnits 3
maxUnits 6

register Spr2020
register Sum2020
register Aut2020
register Win2020

taken CS124
taken CS229
taken CS107
taken CS103
taken CS106X
taken CS106B
taken CS210A

request CS399 # all (1, 9)
request CS221 # Aut,Sum (3,4)
request CS224N # Aut (3, 4)
request CS224S # Spr (2, 4) cannot fulfill due to dependency on cs221 and cs224n
request CS210B in Spr2020 after CS221 # Spr(3, 4) cannot fulfill due to dependency on cs221
request CS223A # Win (3)

The best schedule is:

Quarter	*Units*	*Course*
Spr2020	6	CS399
Sum2020	4	CS221
Aut2020	4	CS224N
Win2020	3	CS223A

Key things to note about this schedule are:
1. CS339 is chosen for first quarter with 6 credits.
2. CS224S and CS210B cannot be fulfilled because they depend on CS221 and CS221 can be taken earliest in Sum2020 but these two courses are only available in Spr2020 (Constraint added from Bulletin and request).
3. CS221 is scheduled in Sum2020 so that CS224N can be scheduled in Aut2020 otherwise the dependency could not have been satisfied.

Since the size of the factors may be as large as n, the tree-width will be at least n in such cases. Tree-width cannot be greater than n since that is the number of variables. Hence, tree-width is n in the worst case.