Problem B. Binary Code

- Solution outline: solve the problem by converting it into an instance of 2-SAT problem

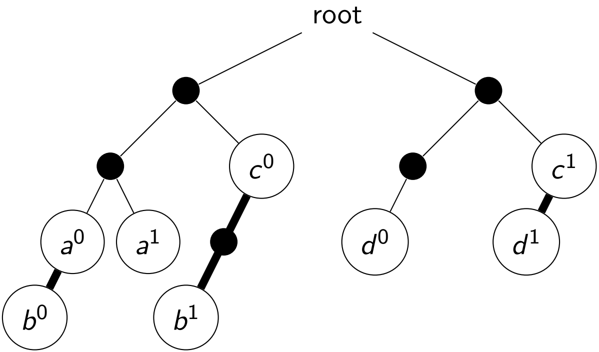
  1. Build a *trie* of the given strings
  2. Define two variables v_i^0 and $v_i^1 = \bar{v}_i^1$ for each word s_i that contains a "?"
     - v_i^0 is *true* and v_i^1 is *false* when "?" is replaced with "0" in s_i
     - v_i^0 is *false* and v_i^1 is *true* when "?" is replaced with "1" in s_i
  3. Create a graph with two nodes for each string. One node for v_i^0, the other for v_i^1
  4. Use the trie to convert binary code constraints into 2-SAT problem instance using *implications*
  5. Use the classical 2-SAT solution algorithm via the graph algorithm to find strongly connected components in *implications graph*

Problem B. Binary Code — Build a trie

- Follow the classic approach, build a *binary* trie
- For strings with "?" add *both* replacements for "?" into a trie
- At the terminal nodes for the string s with "?" put the corresponding variable (s^0 or s^1 depending on replacement)
- At the terminal nodes for the string s *without* "?" put the separate variable T that is always *true*
  - If more than one string without "?" ends at the same node of the trie, the answer is "`NO`"

Problem B. Binary Code — Trie example

- Trie for the first example



a: 00?
b: 0?00
c: ?1
d: 1?0

Implications

a^0 **nand** b^0:

$a^0 \rightarrow b^1$

$b^0 \rightarrow a^1$

c^0 **nand** b^1:

$c^0 \rightarrow b^0$

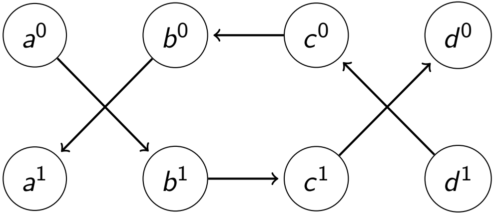
$b^1 \rightarrow c^1$

c^1 **nand** d^1:

$c^1 \rightarrow d^0$

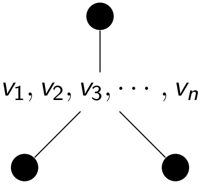
$d^1 \rightarrow c^0$

Problem B. Binary Code — Implications graph example



- Classic 2-SAT algorithm finds the answer or decides that it is impossible
- The sample output assigns *true* to a^0, b^1, c^1, d^0

Problem B. Binary Code — Many terminals at node



- Node in a trie can have many terminals (variables) at one node
- At most one of them can be present in a binary code
- We can express this constraint in $O(n)$ implications using n additional variable pairs
  - Define additional variable r_i to be *true* if and only if at least one $v_j, j \geq i$ is *true*
- Or exclude all v_i and v_j pairs, but return "`NO`" answer when n is more than the depth of this node in a trie plus one