Implement Trie (Prefix Tree)

What to build:

- a TrieNode that stores:
  - children: 26-way pointers/refs (array of size 26 or an unordered_map<char, TrieNode*>)
  - isEnd: bool flag marking end-of-word.
- a Trie with methods:
  - insert(word)
  - search(word)
  - startsWith(prefix)

Design choices:

- Array[26] children
  - fastest & memory-predictable - wastes space if many nodes are sparse.
- Hash/map children
  - saves space for sparse tries - a tiny bit slower due to hashing.

Indices & helpers:

- index for a letter: idx = ch - 'a'
- when walking: if child absent → create for insert, fail for search/startsWith.

Operations (step-by-step):

insert(word)

1. node = root
2. for each ch in word:
   - compute child index/lookup
   - if child missing → create new node and link
   - move node = child
3. after loop: node.isEnd = true

search (word)

1. node = root
2. for each ch in word:
   - if child missing → return false
   - move to child
3. return node.isEnd (must be end of word, not just a prefix).

startsWith (prefix)

1. same walk as search
2. but at the end, don't check isEnd; if you didn't break early, return true

Complexity:

- Let L be the length of the string/prefix.
- insert, search, startsWith are all O(L) time.
- Space: total nodes ≤ sum of inserted word lengths.

Pitfalls to avoid:

- Don't return true in search just because a path exists - must check isEnd.
- Handle empty strings if the platform might pass them (this problem's constraints say length ≥ 1, so you're safe).
- When using array[26], initialize children to null/None (or equivalent).