

## Readings

---

- [Lecture Notes Chapter 24: Tries](#)

## Review

---

A trie is a tree-based data structure that stores strings to support information **retrieval**. Tries are primarily useful when we need to repeatedly query a fixed text because it allows us to pre-process this text such that each subsequent query is fast, offsetting this initial cost of building the trie.

**Standard Trie:** In a standard trie, each root to leaf path corresponds to some string inserted into the trie. If the total length of all strings inserted into the trie is  $n$ , then a standard trie takes  $O(n)$  time to build (using an incremental algorithm) and uses  $O(n)$  space as well.

**Patricia/Compressed Trie:** A compressed trie is a trie where we guarantee that every internal node has at least two children by compressing branches/chains of single-child nodes into a supernode. If the total length of all strings is  $n$  and we have  $s$  strings, then a compressed trie takes  $O(n)$  time to build but only uses  $O(s)$  space, since the tree is now at least as full as a full binary tree (which has  $O(s)$  nodes if it has  $s$  leaves).

**Suffix Trie:** A suffix trie is a trie where the strings are all the suffixes of a string  $S$ . Using an incremental algorithm, we can build a suffix trie in  $O(|S|^2)$  time, but we can actually also do it in  $O(|S|)$  time using Ukkonen's Algorithm; however, the details behind how this works are outside the scope of CIS 121. A compressed suffix trie uses  $O(|S|)$  space.

## Problems

---

### Problem 1

Given a set of  $N$  strings, design an efficient algorithm to find the longest common prefix between any two strings. What is the running time of your algorithm?

### Problem 2

Given some string  $S$ , design an efficient algorithm to find the longest repeated substring. What is the running time of your algorithm?