Supermarket item organizer.

**Challenges:**

* A supermarket will have a variety of items, usually with an identifier(QR code, barcode).
* Each unique item will have an unique code.
* Each set of identical item will have the same code.
* Details of an item must be retrieved using the code.

# Array:

* A large array of an item object can be created.
* The code of each item will be the index where the data presides.
* **Time complexity:** insert, delete, modify, retrieve - O(1)
* **Space complexity:** O(n) (n- number of unique elements)

**Drawbacks:**

* Not scalable, as maintaining huge arrays is not feasible.
* Some codes that are unused lead to data wastage, as the data will be allocated anyway.

# Hash Tables

* Dynamically stores unique values.
* Highly scalable.
* As the codes are already unique, hash collisions are unlikely.
* The Unique code is itself used as the key for lookup.
* **Time complexity**: insert, delete, modify, retrieve - O(1)
* **Space Complexity**: O(n\*k ) (k is size of each node)

Telephone Directory

**Challenges:**

* A phone directory primarily contains a list of Numbers associated with Names.
* Additionally other details like address, email, extra numbers, etc may be present.
* The data must be searchable based on names.
* Entries can be inserted or deleted anywhere in the directory.

# Dictionary

* A dictionary or an associative array may be used which used names as keys, pointing to its corresponding details.
* As names must be unique, the dict data structure seems acceptable.

eg.

|  |  |
| --- | --- |
| name1 | Num1, num2, address, email |
| name2 | Num1, num2, address, email |
| name3 | Num1, num2, address, email |

* **Time complexity**: Search, insert, delete - O(1)
* **Space Complexity**: O(n\*k ) (k is size of each node)

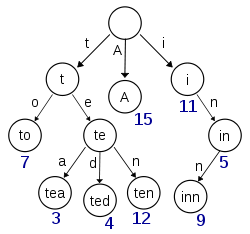
**Drawbacks:**

* A sorted list of all contacts is required. A dictionary stores its values in no particular order. Sorting will be computationally time consuming everytime a list of all contacts is required or a search query is entered.
* So, dictionaries can be considered for a small set of contacts, but is not scalable for large data sets.

# Tree

(Suffix Tree/Trie)

* A suffix tree unlike a binary search tree, does not hold keys associated with the node.
* It stores prefixes as the key, and all the descendants have a common suffix.
* This data structure naturally holds the data in a sorted manner and has extremely efficient query searching.
* Highly scalable compared to Hash tables.
* There is no need for a hash function and hash collision handling.

Eg. 

* **Time complexity**: search, insert, delete - O(L) where L is length of name.
* **Space complexity**: O(N\*K) where N is number of nodes and K is size of each node)

**Drawbacks:**

* Time complexity of lookup is worse compared to hash tables when the data set is small. (low hash collisions)
* If the average name is large, the tree might grow uncontrollably. (Human names usually are small, thus not a huge draw-back)