1 Data structures

1.1 Node

A node consists of a string of symbols called a prefix, an integer value associated with that prefix and two children nodes. It is represented as follows:

$$Node : N(N_l, P, v, N_r)$$

where $P = p_1p_2...$ is the prefix, v is an integer and N_l and N_r are the children nodes. A node can also be empty, in which case it is represented by the special symbol *nil*. A node without any children would then be represented as $N(nil, P, v, nil)$.

1.2 Trie

A trie consists of a special node called the root, an integer representing the number of nodes currently in the trie and another integer representing the maximum number of nodes that the trie can contain. It is represented as follows:

$$Trie : T(R, n, m)$$

where R is the root, n is the current number of entries and m is the maximum number of entries.

2 Functions

2.1 Preliminary definitions

2.1.1 Trie membership

A node is part of a trie if there exists a path from the root to that node.

$$N \in T(R, n, m) \Leftrightarrow \exists RN_1N_2...N_jN$$

such that: $j \leq n - 2$,

$$R(N_1, P, v, N_r) \text{ or } R(N_l, P, v, N_1)$$

and

$$\forall i \in [2, j] : N_{i-1}(N_i, P, v, N_r) \text{ or } N_{i-1}(N_l, P, v, N_i)$$

.

2.1.2 Match length

For the purpose of clarifying the trie specification, we introduce the matchLength function. matchLength takes a node and a prefix as argument and returns the number of symbols in the prefix given as argument that match the symbols in the node's prefix. It is defined as follows:

$$\text{matchLength}(N(N_l, P_N, v, N_r), P) = l \in \mathbb{N}$$

where l is the largest integer such that $p_{N1}p_{N2}...p_{Nl} = p_1p_2...p_l$ and $l \leq min(|P_N|, |P|)$.

2.1.3 Trie condition

To preserve the useful structure of the trie, any node's children have to match the prefix stored in their parent entirely. This is expressed by the trieCondition function, which is defined as follows.

$$\text{trieCondition}(T) =$$

$$\forall N(N_l, P, v, N_r) \in T : \text{matchLength}(N_l, P) = \text{matchLength}(N_r, P) = |P|$$

3 Functions

3.1 initialize

The initialize function takes an integer as argument and returns an empty trie with the specified max number of entries. It is specified as follows:

$$\text{initialize}(m) = T(nil, 0, m)$$

where $m \in \mathbb{N} \setminus \{0\}$.

3.2 update

The update function takes a trie and a prefix-value pair as arguments and inserts a new node in the trie containing the specified prefix and value. If there is already a node corresponding to the specified prefix in the trie, then the value stored in the node is updated. There must be space for at least one more node in the trie given as argument. The update function must not modify or delete any node that does not correspond to the prefix given as argument. It is specified as follows:

$$\text{update}(T(R, n, m), P, v) = T'(R, n', m)$$

where $n < m$, trieCondition(T), trieCondition(T'), $N(\_, P, v, \_) \in T'$,

$$\forall N' \neq N : N' \in T \Leftrightarrow N' \in T'$$

and

$$n' = \begin{cases} n + 1, & \text{if } N(\_, P, \_, \_) \notin E \\ n, & \text{otherwise} \end{cases}$$

3.3 delete

The delete function takes a trie and a prefix as argument and deletes the node corresponding to the given prefix from the trie. There has to be a node in the trie corresponding to the given prefix. The delete function must not modify or delete any node that does not correspond to the prefix given as argument. It is specified as follows:

$$\text{delete}(T(R, n, m), P) = T'(R, n', m)$$

where n' = n-1, trieCondition(T), trieCondition(T'),

$$\forall N(\_, P', \_, \_) \in T' : P' \neq P \text{ and } N \in T' \Leftrightarrow N \in T$$

3.4 lookup

lookup is the main function used to perform the actual longest prefix match. It takes as argument a non-empty trie and a prefix and returns the value stored in the node that best matches the prefix given as argument. It is specified as follows:

$$\text{lookup}(T(R, n, m), P) = v$$

where $n > 0$, trieCondition(T), $N(\_, \_, v, \_) \in T$ and

$$\forall N' \in T, N' \neq N : \text{matchLength}(N', P) \leq \text{matchLength}(N, P)$$