Hypertrie

A *Tire* is a tree structure used to store sequences of characters from an alphabet A. An example of an alphabet is the set of the ASCII codes; sequences, in this case, are strings. A node in a Trie contains potentially one outgoing *edge* for each possible character. Each node in the tree corresponds to a prefix of some sequences of the set, so if the same prefix occurs several times, there is only one node to represent it.

A possible realization of a Trie node is to use an array of pointers of the size |A|. Each pointer can point to either another trie node or null. Each array entry corresponds to a character in the alphabet. As array lookup is computationally constant, looking up sequences in the Trie is fast and requires only O(k) where k is the sequence size. This approach, however, becomes space-inefficient as the size of the alphabet set increases or when it is infinite. An alternative way to represent edges is to maintain a hash table HT: A -> Node whose size will increase as we add distinct keys. Following that, the structure can still be used efficiently to retrieve sequences as accessing keys in a well-implemented sparse hash table is nearly constant.

A fixed-depth trie g is a trie that holds sequences of the same length n. In our context, we call I a key. In that case, a sequence l = <l0, … , lm> of A of length m <= n forms the key prefix. G[l] is defined as the node that is reached from the root node r by walking along the nodes with edges eual to the entries of l. Hence, g[l] could be undefined if no appropriate path exists.

A sequence

Tentris relies on a novel data structure called Hypertire for representing its RDF sparse tensor. Hypertrie is a trie-based data structure. What makes Hypertrie different from normal Trie is that, ….

A fixed-depth Trie [Bra] is seemed to be a straightforward representation for an RDF sparse tensor.

Talk about the efficiency of a trie lookup.