E0 225: Homework 2

Deadline: Oct 22nd, 5pm

Instructions

- All problems carry equal weight.
- Academic dishonesty/plagiarism will be dealt with severe punishment.
- Late submissions are accepted only with prior approval or medical certificate.

1. In this problem you will explore another data structure to answer the dynamic predecessor search problem. Formally, we are have a universe $U = \{0, 1, 2, \ldots, u-1\}$ and a set $S \subseteq U$ of size n. Preprocess S into a data structure so that given a query element $x \in U$, the predecessor of x in S is reported quickly.

The data structure is the following. We will create a hash table containing all the elements of S. Additionally, the hash table will contain every prefix of the binary representation of every element in the set. For example, consider $u = 16$. Then representing each integer in the universe will require four bits. If, say, 14 is in S, then its binary representation is 1110, and hence, all its prefixes $1, 11, 111, 1110$ will be stored in the hash table. Next, we will create a doubly linked list of all the elements of S which are arranged in increased order of their values. Finally, taking inspiration from the lecture on 'augmentation', with each prefix p in the hash table, store a *minimum* and a *maximum*: Among all the elements of S which have p as a prefix, minimum (resp., maximum) of p will be the element with the minimum (resp., maximum) value.

1. What is the space occupied by this data structure?
2. How do you perform *insertion* of a new element in $O(\log u)$ time? (*A short pseudocode or a short description in english will suffice.*)
3. Given a query $x \in U$, the *longest common prefix (lcp)* operation will report that prefix in the hash table which has the longest common prefix with x. For example, if $x = 110111$, and hash table has two strings $y = 110100$ and $z = 111100$, then y has a common prefix of length four with x, while z has a common prefix of length only two with x. How do you perform this operation by querying the hash table only $O(\log \log u)$ times? (*Hint: Every element in U in binary representation requires $O(\log u)$ bits*). First, give a few lines overview of your algorithm. Then follow it up with a short pseudocode.[1]
4. Assume that the lcp operation can be performed in $O(\log \log u)$ time. Using the lcp operation, how do you perform predecessor search operation in $O(\log \log u)$ time?
5. (*Will not be graded.*) This solution is still unsatisfactory, since the insertion time is $O(\log u)$. Use the grouping trick from the lecture, where groups of size (roughly) $\Theta(\log u)$ are formed. A binary search tree is built for each group. From each group, a *representative* value is chosen and the above data structure is built on *only* the representative values. The representative value of the ith group is at least as large as the largest element in the ith group, and it is smaller than every element in the $(i+1)$th group. The representative values need not belong to S.
    - Can you see why the space is reduced to $O(n)$?

[1] Please spare the TA from having to read long essays :)

- How to perform lcp and predecessor search operation still in $O(\log \log u)$ time?
- How do we handle insertions now, so that the amortized insertion time is reduced to $O(\log \log u)$?

Conceptually, it might help to think of the data structure as a trie data structure with fanout two. The data structure is built based on the binary representation of the elements in the U. The elements of S are at the leaf level. The internal nodes represent the prefixes being stored in the hash table.

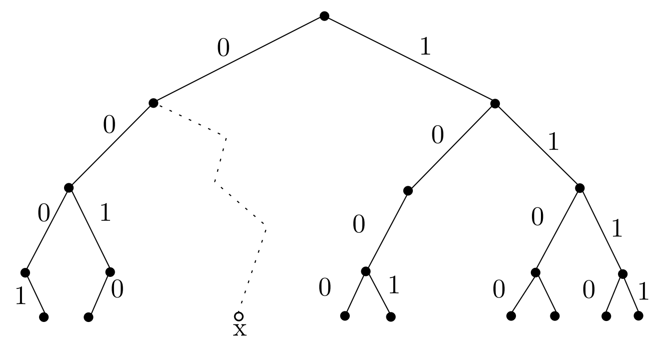

Figure 1: A trie with $u = 16$ and $S = \{1, 2, 8, 9, 12, 13, 14, 15\}$, For a query element x, its predecessor is 2 and its lcp is the prefix 0.