11/21/2016

ALGORITHMS

HW-10

Kiran Shettar
UML ID - 01605800

Problem 1: 32.1-2 (pg 989)

Now from the given problem, we know that a single occurance of 'P' cannot be in T i.e occurance of 'P' in 'T' cannot overlap with one another. So there is no need to double check the way NAIVE-STRING-MATCHER algorithm does. If we find an occurance of P_k in the text followed by a non-match, we can increment 's' by 'k' instead of 1.

This way we can accelerate NAIVE-STRING-MATCHER to run in $O(n)$ on an n-character text T.

Problem 2: 32.2-3 (pg 994)

Firstly we have to maintain a hash table of the pattern & computing running hashes of the text. If we update the hash @ each step then the running time will be $\Theta(m)$ as the number of entries which are both entering & leaving the ~~same~~ hashe window would be 2m, & we have

(2)

to keep on seeing them entering & leaving. In order to compute this hash, we will be giving each entry of the window a power of 'd' that is unique to its position. The entry in row 'i' & column 'j' would be multiplied by d^{m^2-mi+j}. Then we move to our right. Then we multiply the value of the hash by 'd', substract off the scaled entries that were in the left column, and add in the entries that are in the right column, also appropriately scaled by what row they are in. Similarly for shifting the window up/down. Like this we can extend the Rabin Karp method to the problem of ~~searching~~ looking for a given mxm pattern in an nxn arrays of characters.

Problem 3: 32.3.-3 (pg 1002).

The state transition function looks like a straight line, with all other edges going back to either the intial vertex (if it's not the first letter

letter of the pattern) or the second vertex (if it's the first letter of the pattern). If it were to go back to any other later state, that would mean that some suffix of what we had constructed so far (which was the prefix of P) was a prefix of the copy of P that we are next trying to find.

Problem 4 : 32.3-5 (pg 1002).

To create a DFA that worked with gap characters, construct the DFA so that it has $|P| + 1$ states. Let 'm' be the number of gap characters suppose that the positions of all the gap characters within the pattern 'P' are given by g_i, and let $g_0 = 0$, $g_m = |P| + 1$. Let the segment of pattern occuring after gap character 'i' but before the $i+1$ gap character be called P_i. Then, we will imagine that we are trying to match each of these patterns in sequence, but if we have trouble matching some

particular pattern, then we cannot undo the success we enjoyed in matching earlier patterns.

If we have $(Q_i, q_{i,0}, A_i, \Sigma_i, \delta_i)$ is the DFA corresponding to the pattern P_i. Then we will construct our DFA so that $Q = \bigcup_i Q_i$, $q_0 = q_{0,0}$ $A = A_{m+1}$, $\Sigma = \bigcup_i \Sigma_i$ and the δ (transition). Consider that we are at state $q \in Q_i$ and see the character a, if $q \notin A_i$ then we just go to the state given by transition of $\delta_i(q, a)$. If however we have that $q \in A_i$ then $\delta(q, a) = \delta_{i+1}(q_{i+1}, 0, a)$. Like this we can build a finite automaton that can find an occurance of 'P' in a text 'T' in $O(n)$ matching time, where $n = |T|$.