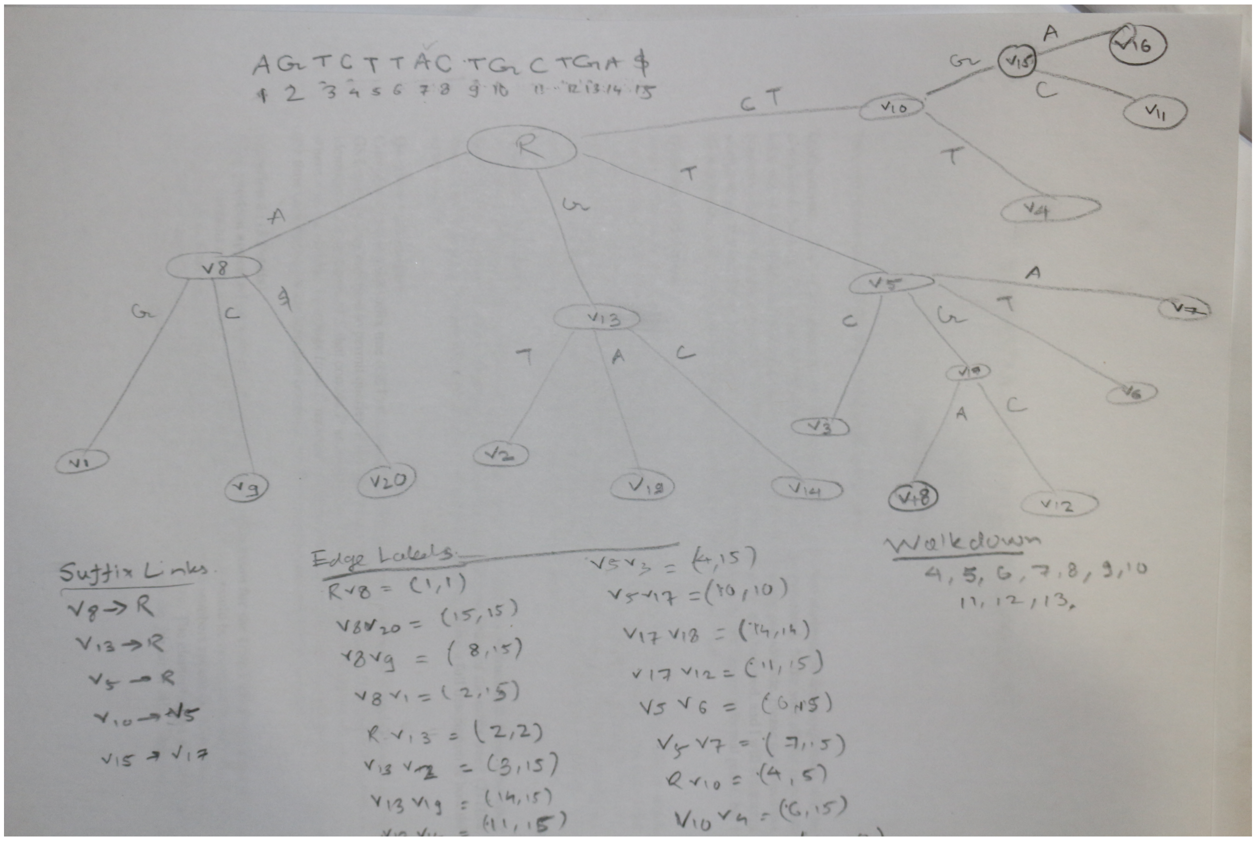
MATH-578A: Homework # 1

Due on Tuesday, March 10, 2015

Saket Choudhary
2170058637

Contents

Question # 1



Suffix Links.

$v_8 \to R$

$v_{13} \to R$

$v_5 \to R$

$v_{10} \to v_5$

$v_{15} \to v_{17}$

Edge Labels:

$Rv_8 = (1,1)$	$v_5v_3 = (4,15)$
$v_8v_{20} = (15,15)$	$v_5v_{17} = (10,10)$
$v_8v_9 = (8,15)$	$v_{17}v_{18} = (14,14)$
$v_8v_1 = (2,15)$	$v_{17}v_{12} = (11,15)$
$Rv_{13} = (2,2)$	$v_5v_6 = (6,15)$
$v_{13}v_2 = (3,15)$	$v_5v_7 = (7,15)$
$v_{13}v_{19} = (14,15)$	$Rv_{10} = (4,5)$
[illegible] $= (11,15)$	$v_{10}v_4 = (6,15)$

Walkdown

4, 5, 6, 7, 8, 9, 10

11, 12, 13,

Question # 2

Space required to store the nodes of suffix tree : O(n)
With edge label compression, each ege can be stored with two integers at each node denoting the start and end positions of the substring collapsed.
Root: Stores nothing except at least two outgoing pointers. Each internal node stores two *integers* representing the start and end position so that the total size taken by each internal node is $2 * sizeof(int)$
Number of internal nodes $= n - 1$. Hence total size occupied by nodes $= (n-1) * 2 * (sizeof(int*))$. The leaves store just one integer representing the start position of the suffix, so occupy $n * sizeof(int*)$ space. The edges are represented by pointers, so for a total of $2n - 1$ nodes, there are $n - 2$ edges for the internal nodes, each taking $sizeof(int*) + sizeof(int*)$ for representing the two integers at each node. Whike the pointers to the leaves takeup $n * sizeof(int*)$. In total: Internal nodes: $(n-1) * 2 * sizeof(int)$
Leaves: $n * sizeof(int)$
Internal Edges(Pointers): $(n-2) * sizeof(int*)$
Terminal Edges(Pointers): $n * sizeof(int*)$
Total $= 3(n-1) * sizeof(int) + (2n-2)sizeof(int*) = 3(n-1) * (8) + (2n-2) * (64) = 136(n-1)$

For a GST, on the leaves, we need to store the index of string that suffix came from, so an additional $n * (sizeof(int*)) + n * (sizeof(int))$, totaling $136(n-1) + 64n + 8n = 208n - 136$

continued on next page...

Question # 3

At each internal node store the identity of the leaves below it whether they come from $S_1, S_2..., S_k$ etc. This can be done using depth first traversal occupying $O(kn)$ space and $O(kn)$ in time. Since each internal node has atleast two children, the depth first traversal in a subtree is bounded by $O(kn)$.
Construction of a GST is linear. It would require concatenating strings separated by unique delimiters($1$2$3...) and then follow suffix tree construction algorithm. Once the suffix tree has been constructed, a DFS from the root to leaves can then be used to store at each internal node which all string's suffixes are represented in the leaves below it. This is linear time too. The asymptotic space complexity is $O(kn)$ since each node can store at a maximum of $k+2$ values where k represents number of different strings. Finding occurences of P would involve tree traversal untill a fall off occurs at node v after m matches and the string ids stored at v can be retrieved in constant time.

Question # 4

We will assume n is a of type $2^k - 1$. Make a $O(n)$ query for a LSB of the n numbers. The number of 1 and 0 should ideally be same were all numbers present. Whichever occurence is smaller(0 or 1)($floor(\frac{N}{2})$), we do this for next $\frac{N}{2}$, then next $\frac{N}{4}$ and so on (total $O(2N)$) at each point storing which was lower $0 or 1$ and then recontruct the missing sequence.
Example:
$n = 2^3 - 1 = 7$ so numbers from 0 to 7 should have been present. Let's say 5 is missing. 000
001
010
011
100
110
111

First quesry on 7 digits gives 4 zeroes and 3 ones. So LSB of missing number os 1. Now repeat for next significant bit for 3 numbers: 001
111
111

2 ones and 1 zeroes, so the next bit is 0. Finally: 001
1 zero and 0 ones. So missing bit is 1 Missing number: 101

Question # 5

Hash table sizes are often prime numbers to minimise probability of collisions of unlike quantities. Prime number modules ensures the number being divided go to maximum possible buckets(because the dividend and divisor will often be coprime). In the context of Rabin Karp, choosing a suitable prime is essential to prevent false positives, which would increase running time.

continued on next page. . .

Question # 6

Identifying longest substring starting at each position of T that occurs in P. This requires creating a suffix tree for the pattern and not the text. Consider first m characters of text T and do a traversal of the suffix tree of pattern P. The last visited node's node edge depth will give the length of longest substring match in $T[1..m]$ to some corresponding substring in P. Now consider finding sucha substring that starts from position 2 of T i.e $T[2..m]$. This would now make use of the suffix link property. If the deepest possible visited during first step has path label $x\alpha$ then, the prefix of $T[2..m+1]$ is α which infact is just the suffix link and this saves explicit character matches. Explicit matches are now performed from the suffix link node.

continued on next page...