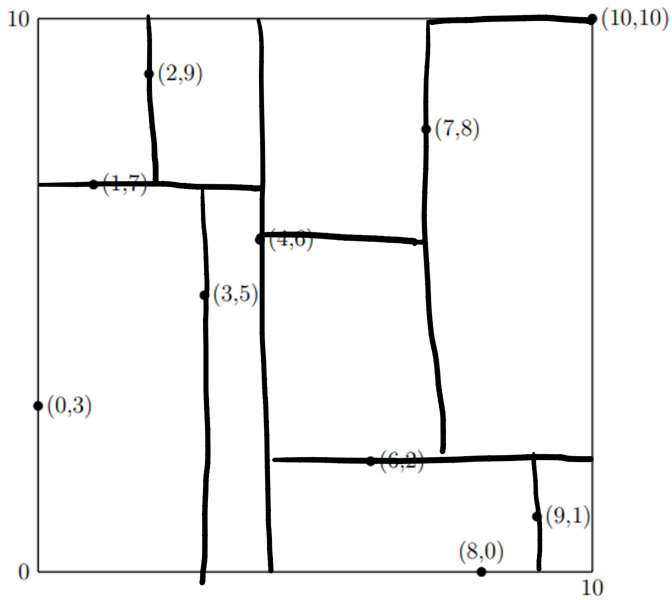
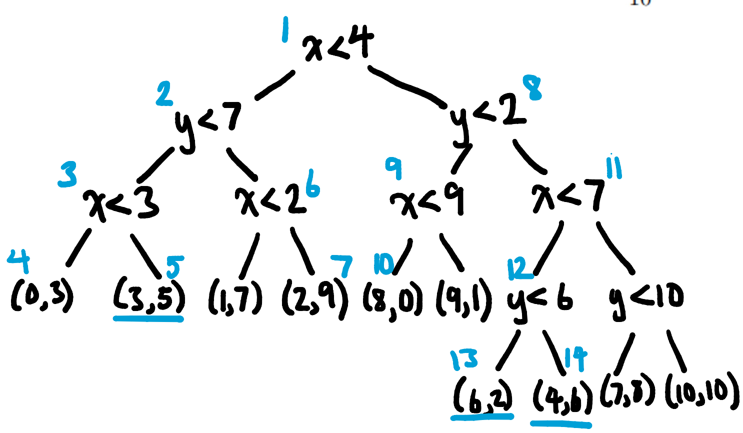
CS 240 S2017 Final Practice

T/F a) False b) True c) False d) False e) N/A f) False g) False h) True i) False j) True

MC 1. d 2. N/A 3. a 4. c 5. b 6. b 7. c 8. c

KD





KMP

0	1	2	3	0

J	W	X	J	J	W	J	J	W	J	W	J	W	J	X	J	W	X	J	J
J	W	J																	
		J																	
			J	W															
				J	W	J	W												
						(J)	W												
							J	W	J	W	J	X							
									(J)	(W)	(J)	W	J	X					

Hash a)

0	0	- 2
1	1	- 3
2	6	- found
3		
4		
5		
6	13	- 1

b) It'll be the exact same table :P

Huff a) 001 000 011 1010 1011 100 010 11 010 010
pswd: lull

b) u has the shortest code, but is not the most frequent character in the string.

RLE a) 1 1 2 1 4 1 8 8 1 3 1 1 1

0 1 1 010 1 00100 1 0001000 00111 1 011 1 1 1

b) $\frac{32\lceil \log 2\rceil}{32\lceil \log 2\rceil} = 1$

c) 1 1 1 1 0001110 00111 0001100 011 00100 1
1 1 1 14 7 12 3 4 1

1010000000000000001111110000000000000011100001

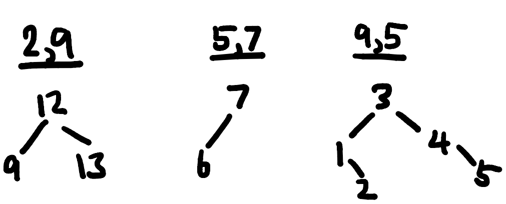
LZW 68-65-82-75-95-128-78-95-66-129-75-83-132-65-78-75

DARK_DAN_BARKS_DANK

128: DA
129: AR
130: RK
131: K_
132: _D
133: DAN
134: N_
135: _B
136: BA
137: ARK
138: KS
139: S_
140: _DA
141: AN
142: NK

SM Automata Not covered :P

Range Trees



Boundary nodes: (7, 10), (4, 8), (2, 9), (1, 13)
Outside nodes: (9, 5)
Inside nodes: (3, 12), (5, 7), (6, 6)

Tries

We range search the trie with query [b1, b2]. Assume, without loss of generality, that b1 < b2. We find the search path P1 for b1, and the search path P2 for b2. We have that a node is an inside node if it is to the right of P1 and to the left of P2. The inside nodes whose parents are boundary nodes are top inside nodes. If there exists a top inside node, return False; otherwise return True.

If b1 and b2 are consecutive, then the range-search would have no other keys in the range aside from b1 and b2, so there would be no inside nodes. If b1 and b2 are not consecutive, then there exists at least one other key, b3, which lies between b1 and b2. All boundary nodes are internal nodes except for the two nodes at the ends of the paths P1 and P2, which correspond to the leaves containing b1 and b2. Since b3 is in the lexicographic range, it must be a leaf as an inside node.

We analyze the runtime. The boundary involving b1 has at most |b1| nodes, while the boundary involving b2 has at most |b2| nodes. It takes constant time to check for the existence of top inside nodes. Hence the runtime is O(|b1| + |b2|).

Order

a) g(n) = n
f(n) = n^2

b) We prove the statement. If $f(n) \in O(g(n))$, then there exists constants $c_1 > 0$ and $n_0 \geq 0$ such that $|f(n)| \leq |c_1(g(n))|$ for all $n \geq n_0$. Since both f (n) and g(n) map positive integers to positive integers, we can drop the absolute value signs and say $f(n) \leq c_1(g(n))$ for all $n \geq n_0$. We can define a constant c_2 as the maximum of g(n) in the range $(0, n_0]$ plus c_1. Since both functions map positive integers to positive integers, c_2 is greater than or equal to maximum of g(n) in that interval. This means that in the interval $n \in (0, n_0]$, we have $g(n) \leq c_2 f(n)$. With this same logic, also have $c_2 \geq c_1$, and hence we still satisfy $g(n) \leq c_2 f(n)$ for all $n \geq n_0$. Since we have $f(n) \leq c_2(g(n))$ for all $n \geq n_0$ and for $n \in (0, n_0]$, we have $f(n) \leq c_2(g(n))$ for all $n > 0$. Hence by definition, $g(n) \in$ Onion(f (n)).

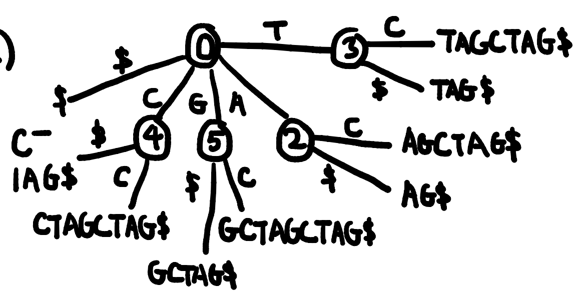
BWT

($, 8)	(A, 9)	(A, 12)	(E, 0)	(E, 2)	(E, 11)	(L, 5)	(P, 1)	(P, 6)	(P, 7)	(R, 13)	(S, 4)	(S, 10)

PAPERSPLEASE$

Suffix Trees

a)



b) An internal node has at least two leaves, so its substring is repeated at least twice. We find the deepest internal node in the suffix tree.

Range Query

We create a BST, where the key is the index of the array, and each node stores the value, the minimum value in its subtree, and the maximum value in its subtree. To execute a range query, we use BST::rangeQuery to find the minimum value of all nodes in range. We do the same for the maximum value of all nodes in the range. We subtract the two and return this value.

```
maxDiff(i, j) {
	return rangeMax(i, j) - rangeMin(i, j)
}

rangeMax(i, j) {
	P1 = path along binary search for i
	P2 = path along binary search for j
	return max(values in P1, values in P2, subMax of topmost inside nodes)
}

rangeMin(i, j) {
	P1 = path along binary search for i
	P2 = path along binary search for j
	return min(values in P1, values in P2, subMin of topmost inside nodes)
}
```