University of Waterloo
CS240 Spring 2022
Assignment 5 - Marking Scheme

Marking Deadline - Friday, July 29th, at 7:00pm

Problem 1 Max-x in kd-trees [5 marks]

Give an algorithm to find the point with the maximum x-value in a 2D kd-tree, and analyze its complexity. For maximum credit, your algorithm must run in $o(n)$ time, where n is the number of points in the kd-tree. For simplicity, you may assume that n is a power of 4. If the run-time $T(n)$ of your algorithm satisfies a recurrence relation that has already been seen in class, you can take for granted the corresponding growth rate for $T(n)$, without giving the proof.

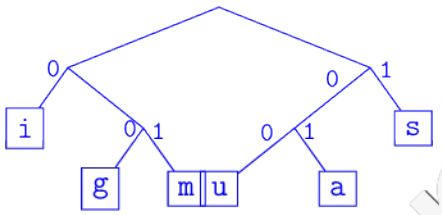
Solution: For any node that corresponds to a vertical line (compares x-coordinates), only check the right child. For all the other internal (non-leaf) nodes, check both children, compare the results, and output the smaller one. For $d = 2$, we have the following recurrence relation for every two steps of our recursion. $T(n) = 2T(n/4) + \Theta(1)$. Hence, this algorithm runs in $O(\sqrt{n})$.

Problem 2 Huffman Coding [2+3 = 5 marks]

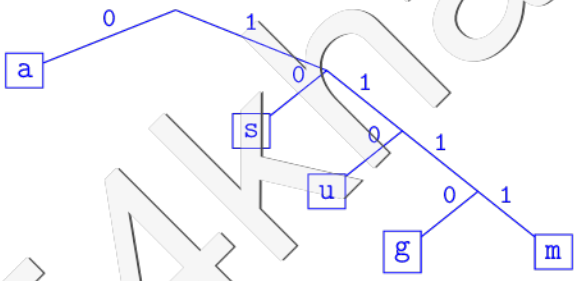
a) Trace the algorithm to build the Huffman trie for the string `mississauga`, and name the result T_1. Repeat the process for the string `massasauga`, and name the result T_2.

To break ties for tries of the same frequency, make the trie whose characters have a smaller ASCII sum the left child of the new root (that is, make it follow the "0" label).

T1



T2



b) Compare the compression rates of the tries from part (a) for encoding the string `massasagua`.

symbol	freq.	length in T1	all in T1	length in T2	all in T2
a	4	3	12	1	4
g	1	3	3	4	4
m	1	3	3	4	4
s	3	2	6	2	6
u	1	3	3	3	3
sum	10		27		21

For both source alphabets, we need $3 = \lceil \log 5 \rceil = \lceil \log 6 \rceil$ bits. Hence, without the encoding we would need 30 bits. With encoding using T1, the compression ratio is $\frac{27}{30}$ and using T2 the compression ratio is $\frac{21}{30} = 70\%$.

Problem 3 Run-Length Encoding [2+1+3+2 = 8 marks]

Suppose we want to encode a ternary string, where each symbol is 0, 1, or 2, to a ternary string using RLE.

a) What do you need to change in the RLE from the slides to make it work for a ternary string?

Solution: since the alternating pattern between 0's and 1's will not happen here, we need to find a way to represent the symbol in the next run.

- One way is to start from a given symbol, and switch from 0 to 1, from 1 to 2, and from 2 to 0.
- Alternatively, we can represent the symbol for each run.
- A third alternative is to use runs of one symbol to indicate the type of the next run (2 for 0, 22 for 1, 222 for 2), and runs of the other two symbols to indicate the length of the run.

Note: If the alternating pattern is used (switch from 0 to 1, 1 to 2, and 2 to 0) then runs of zero are represented. In this case, the Elias gamma codes used should be changed to start with $\lfloor \log(k+1) \rfloor$ leading zeros.

b) What are your ideas for making sure you are optimizing the compression rate?

Solution: The first approach forces us to represent runs of length zero, but that is inefficient. So we just use a symbol to indicate the type of the run. Also, we use all three digits to represent the runs, so we will need fewer symbols to represent the run (this is an acceptable solution.)

c) Present your algorithm as a pseudocode.

```
encoding(S, C)
S: input-stream of bits, C: output-stream
1.    while S is non-empty do
2.        b ← S.top();
3.        for i : 0 → b do
4.            C.append(2)          // represent the symbol
5.        k ← 0                    // length of run
6.        while (S is non-empty and S.top() = b) do
7.            k; S.pop()
              // compute and append binary representation
8.        K ← empty string
9.        while k > 0
10.           K.prepend(k mod 2)
11.           k ← ⌊k/2⌋
12.       C.append(K) ;
```

d) What is the compression rate of your encoding scheme for the following ternary string?

012001122000111222000011112222000000000000001111111111111111222222222222222222.

$$C = 21221222121022102221021122112222112100221002221002111022100002221000 1$$

$$\frac{|C|}{|S|} = \frac{68}{77}$$

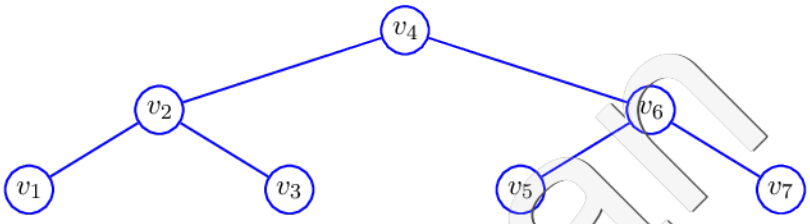
Problem 4 Range Trees [3+4+4=11 marks]

a) Draw a 2-dimensional range tree of minimal height for the following set of average assignment and midterm grades:

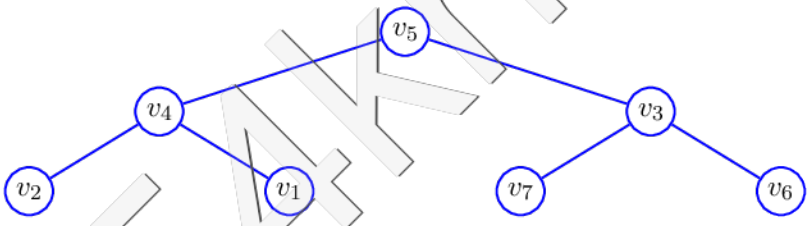
$$\{(81, 70), (85, 68), (88, 88), (94, 68.5), (97, 75.5), (97.5, 100), (100, 80)\}$$

$$v_1 = (81, 70), v_2 = (85, 68), v_3 = (88, 88), v_4 = (94, 68.5),$$
$$v_5 = (97, 75.5), v_6 = (97.5, 100), v_7 = (100, 80)$$

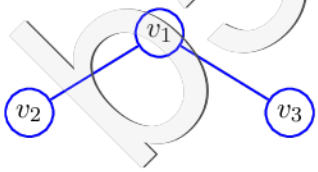
T:



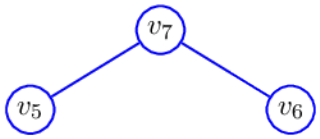
$T(v_4)$:



$T(v_2)$:



$T(v_6)$:



b) Assume that we have a set of n numbers (not necessarily integers) and we are interested only in the number of points that lie in a range rather than in reporting all of them. Describe how a 1-dimensional range tree (i.e., a balanced BST) can be modified such that a range counting query can be performed in $O(\log n)$ time (independent of s). Briefly justify that your algorithm is within the expected run-time.

Solution: For each node, keep the number of its descendants, minimum descendant, and maximum descendant. For each node, if minimum and maximum descendants are in the range, count the number of descendants, otherwise, follow the regular range search algorithm, and add up the results from the sub-problems.

Analysis: For all boundary (blue) nodes, we may include the node itself, but there are $O(logn)$ of those. For each topmost inside node, we only report the number of its descendants, and there are at most $O(logn)$ of those. The comparison of min and max descendants against the bounds is doable in $O(1)$. Hence the whole algorithm is doable in $O(logn)$.

c) Next, consider the 2-dimensional case where we have a set of n 2-dimensional points. Given a query rectangle R, we only want to find the number of points inside R, not the points themselves. Explain how to modify the Range Tree data structure discussed in class such that you can answer any of these counting queries in $O((\log n)^2))$ time. Briefly justify that your algorithm is within the expected run-time.

Solution: Similar to the approach from part (b). The only difference is that the nodes in each tree only store the min, max, and number of descendants based on the the corresponding dimension. Here, each top most inside node needs $O(\log n)$ time (similar to the analysis above) to find the descendants in range, and there are $O(\log n)$ of them, so the entire algorithm is in $O((\log n)^2)$

m	i	s	s	i	s	s	a	u	g	a	m	i	s	m	i	s	s	i	s	s	i	p	p	i	m	i	l	l
X																												
	i	s	s	i	s	s	X																					
				(i)	(s)	(s)	X																					
							X																					
								X																				
									X																			
										X																		
											X																	
												i	s	X														
														X														
															i	s	s	i	s	s	i							

Table 2: Table for KMP problem (you may need to add rows, or delete some).

Problem 5 KMP [3+4=7 marks]

a) Compute the failure array for the pattern $P =$`ississi`.

Solution: $F = [0, 0, 0, 1, 2, 3, 4]$

b) Show how to search for pattern

$P =$`ississi`

in the text

$T =$`mississaugamismississippimill`

using the KMP algorithm. Indicate in a table such as Table 2 which characters of P were compared with which characters of T. Follow the style in the example on slide 16 in Module 9.

- Place each character of P in the column of the compared-to character of T.
- Put brackets around the character if an actual comparison was not performed.
- Use a new row when sliding the pattern.
- You may not need all space in the table.

Problem 6 Boyer-Moore [1+3+3+3+5=15 marks]

a) Construct the last occurrence function L for pattern $P = ramammam$ where $\Sigma = \{a, k, m, r\}$.

c	a	k	m	r
L(c)	6	-1	7	0

b) Trace the search for P in $T = ramaamamkmamramammam$ using the Boyer-Moore algorithm. Indicate in a table such as Table 3 which characters of P were compared with which characters of T. Follow the example on slides 23-24 in Module 9.

- Place each character of P in the column of the compared-to character of T.
- Put brackets around the character if they are known to match from the previous step (similar to the examples in the slides).
- Use a new row when sliding the pattern.
- You may not need all space in the table.

r	a	m	a	a	m	a	m	k	m	a	m	r	a	m	a	m	m	a	m
				m	m	a	m												
								m											
													m	m	a	m			
																a	m		
																		m	
												r	a	m	a	m	m	[a]	m

Table 3: Table for Boyer-Moore problem.

c) For any $m \geq 1$ and any $n \geq m$, give a pattern P and a text T such that the Boyer-Moore algorithm looks at exactly $\Theta(n/m)$ characters. Justify your answer.

Sample Answer

Let P be a pattern of length m using the alphabet $\Sigma = a, w, x, y$ and T be a string comprised of $\mathcal{A} \backslash \Sigma$, where $\mathcal{A}$ is the set of lower-case English alphabets.

Since we will always have a character mismatch at the very first comparison using the bad-character heuristic, we will shift by m each time, thereby leading to $\Theta(n/m)$ comparisons in total.

d) For any $m \geq 1$ and any $n \geq m$ that is a multiple of m, give a pattern P and a text T such that the Boyer-Moore algorithm looks at all characters of the text at least once and returns with failure. Justify your answer.

Answer

Sample Pattern and Text: P = *arun* ; T = $([\hat{}a]run)^{\frac{n}{4}}$ (e.g. brun, crun, ... zrun). (Note that since n must be a multiple of m and here $m = 4$, $\frac{n}{m} \in \mathbb{Z}$).

In this case, the comparison will always fail at the first character of the pattern, but since comparisions are done in reverse, the other 3 would already have been looked at. Therefore, the pattern would be shifted by at least 1, and the same process will repeat until all characters have been looked at.

General case: Let $n = km$ for some $k \in \mathbb{N}$. Then, for an arbitrary pattern P, a text $T = ([\hat{}P[0]]P[1...m-1])^k$ would yield $O(n)$ comparisons, and in particular would look at every character in the text.

e) A number of heuristics can be used with Boyer-Moore to reduce the number of comparisons performed between P and T. Suppose we use Boyer-Moore with only the Peek heuristic. The Peek heuristic states that if $P[j] \neq T[i]$ and $P[j-1] \neq T[i-1]$ then the next location to search for P at is $T[i+m-1]$. Show that the Peek heuristic may fail to find P in T, i.e., find a pattern P, and a text T containing P, such that Peek fails to find P in T. Justify your answer in a short paragraph.

Sample Answer (with reverse heuristic)

P = iguana
T = sickiguanaisrecovering

Justification: First, T[5] and P[5] (g and a) will be compared. They don't match, so we'll look at T[4] and P[4] (i and n). Again, we have a mismatch, so we jump to T[5 + (6 − 1)] = T[10]. Here we compare T[10] and P[5] (i and a), followed by T[9] and P[4] (a and n) – both of which lead to mismatches. We then move to T[10 + (6 − 1)] = T[15]. By this point, we have skipped the only occurrence of the pattern in T, thus showing that the Peek Heuristic will not always find the pattern in the text.

Sample Answer (without reverse heuristic)

P = abcd
T = dabcdabc

Justification: First, T[1] and P[1] (a and b) will be compared. They don't match, so we'll look at T[0] and P[0] (d and a). Again, we have a mismatch, so we jump to $T[4 + (1 - 1)] = T[4]$. Here we compare T[4] and P[1] (d and b), followed by T[3] and P[0] (c and a) – both of which lead to mismatches. We then move to $T[4 + (4 - 1)] = T[7]$, which is the end of the text. By this point, we have skipped the only occurrence of the pattern in T, thus showing that the Peek Heuristic will not always find the pattern in the text.