University of Waterloo
CS240 - Winter 2021
Assignment 5

There are written questions and a programming question in this assignment.

- the deadline for **all written questions** (1 to 5) is Wednesday April 7, 5pm;
- the deadline to submit your file `encode.cpp` `decode.cpp` **from Problem 6** is Wednesday April 14, 5pm.

The integrity of the grade you receive in this course is very important to you and the University of Waterloo. As part of every assessment in this course you must read and sign an Academic Integrity Declaration (AID) before you start working on the assessment and submit it **before the deadline of April 7** along with your answers to the assignment; i.e. **read, sign and submit A05-AID.txt now or as soon as possible**. The agreement will indicate what you must do to ensure the integrity of your grade. If you are having difficulties with the assignment, course staff are there to help (provided it isn't last minute).

The Academic Integrity Declaration must be signed and submitted on time or the assessment will not be marked.

Please read `http://www.student.cs.uwaterloo.ca/~cs240/w21/guidelines/guidelines.pdf` for guidelines on submission. **Each written question solution must be submitted individually to MarkUs as a PDF** with the corresponding file names: a5q1.pdf, a5q2.pdf, ...

It is a good idea to submit questions as you go so you aren't trying to create several PDF files at the last minute. **Remember, late assignments will not be marked but can be submitted to MarkUs after the deadline for feedback if you email cs240@uwaterloo.ca and let the ISAs know to look for it.**

Problem 1 A variant of 2d range search [4+2 marks]

Consider the following variant of two-dimensional range reporting queries. We keep a set of two-dimensional points P in a data structure. All points have positive coordinates. For a query range $Q = [a, b] \times [0, c]$, we must find some point p in $P \cap Q$ or report NULL if the intersection is empty. In other words, we must find one point p such that $p.y \leq c$ and $a \leq p.x \leq b$ (we do not need all of them; only one!)

a) Describe a data structure for queries as described above, that uses space $O(n)$. Give an algorithm to build the data structure, with its complexity.

b) Give the search algorithm and analyze its runtime. For maximal credit, search should take time $O(\log(n))$.

As usual, justify your answers. You do not need to explain how to update the data structure.

b	a	c	b	a	b	a	a	b	a	b	a	c	b	a	c	b	a	c	a

Table 1: Table for KMP problem.

Problem 2 Min-x in kd-trees [5 marks]

Give an algorithm to find the point with the minimum x-value in a kd-tree, and analyze its complexity. For maximum credit, your algorithm must run in time $O(f(n))$, where n is the number of points in the kd-tree, for some function $f(n)$ that you have to give us, and which must be $o(n)$. For simplicity, you may assume that n is a power of 4.

If the runtime $T(n)$ 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.

Problem 3 KMP [5+5 marks]

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

b) Show how to search for pattern $P =$`bacbac` in the text $T =$`bacbabaababacbacbaca` using the KMP algorithm. Indicate in a table such as Table 1 which characters of P were compared with which characters of T. Follow the example given in class: place each character of P in the column of the character of T we compare it to and put brackets around the character if an actual comparison was not performed. You may not need all space in the table.

Problem 4 Suffix trees and suffix arrays [4+4 marks]

The two questions are independent.

a) Give an algorithm (in pseudo-code) that takes as input the suffix tree of a string S of length n, and returns its suffix array. To write the pseudo-code, you can assume that

- each node r in the suffix tree holds a boolean r.`is_leaf`
- at each internal node r of the suffix tree, the list of children is implemented as an array $r.A = [(c_1, r_1), \ldots, (c_s, r_s)]$, where s is the number of children, $c_1 < \cdots < c_s$ are the characters that label the outgoing edges (in increasing order), and $r_1, \ldots, r_s$ are references to the corresponding children.
- each leaf r (which corresponds to a suffix $S[i..n]$, for a certain index i) only stores the corresponding index i as r.`index`.

Analyse the runtime; for full credit, your algorithm should run in time $\Theta(n)$.

b) Give an algorithm (in pseudo-code) that takes as input the suffix array of a string S of length n and a string P of length $m \leq n$, and returns the number of occurences of P in S.

Analyse the runtime; for full credit, your algorithm should run in time $O(m \log(n))$.

Problem 5 Huffman encoding [5+5 marks]

The two questions are independent

a) Trace the algorithm to build the Huffman tree for the string `bananamuffinbun` and give the coded text. For conventions on how to break ties, see the discussion in the last problem.

b) Huffman's algorithm sends the decoding tree along with the coded text; the decoder has to reconstruct it. There are several ways to achieve this; in this problem, we consider one of them.

In this approach, the encoder does a traversal of the Huffman tree; at any internal node, we visit the left child before the right child. When we reach a leaf, we output the character at this leaf, together with the *depth* of the leaf (that is, the length of the corresponding codeword). On the example p.15 in the lectures, this would give the list $[G, 3], [Y, 3], [E, 2], [R, 2], [N, 2]$.

Give an algorithm of cost $O(n \log n)$ that reconstructs the Huffman decoding tree from such a list, where n is the number of leaves in the tree (justify your answers, as usual). Hint: use techniques similar to the construction of the Huffman tree from frequencies.

Problem 6 Huffman encoding [15 marks]

In this problem, you are going to implement Huffman's encoding and decoding algorithms. You will write them in two files, `encode.cpp` and `decode.cpp`.

In `encode.cpp`, the `main` function reads one line from `cin`; this is the text you have to encode. It will output two lines:

1. The Huffman tree as a sequence of the form $c_1 d_1\ c_2 d_2\ \ldots\ c_s d_s$, as in the previous problem. The c_i's are characters, the d_i's their respective depths. There is no space between c_i and d_i; each d_i is followed by a whitespace (even the last one).

2. The coded text, as a sequence of 0's and 1's (no whitespace).

We give you a starter file that contains the definition of a class of binary trees (which you'll probably have to complete) and `main` function (which you have to complete as well).

In `decode.cpp`, the `main` function reads two lines: these are supposed to be the output of `encode`. t will output the decoded text on one line. Overall, with input `input.txt`, doing `./encode < input.txt | ./decode` should give you back the input text.

A few comments:

- Do not expect to actually compress your inputs, as we print the coded text as an actual sequence of 0's and 1's. We should encode them into bytes to see an actual compression, but this is harder to debug.

- We recommend you use `unsigned char`'s as your alphabet, so as to be sure they will be cast to integers in $0, \ldots, 255$.

- The encoding algorithm uses a heap. In order to standardize your answers in the presence of ties, use the following conventions:

  - the heap operations should be implemented as in module 2
  - when you populate the heap initially, visit all (unsigned) characters from 0 to 255 in this order.
  - when you remove T_1, then T_2 from the heap, make T_1 the left child and T_2, the right child of the tree you construct.

You cannot use STL queues. You can use vectors. We give you a sample pair input / output (coded text), together with a mystery output file, that you should be able to decode.