Note: Any solution to an algorithm design question MUST contain the following

four sections:

(1) Problem statement. A clear unambiguous statement of the problem to

be solved, which includes the input, the output, and the object function

with the constraints.

(2) Algorithm description. A clear, unambiguous description of the algo-

rithm.

(3) Correctness proof. A convincing mathematical argument that the algo-

rithm described solves the computational problem described.

(4) Time analysis. A time analysis of the algorithm, up to order, in terms of

all relevant parameters.

You may use any algorithms and data structures from class.

1. Ukkonen’s algorithm

(I) Formalize the pseudocode for the Ukkonen’s algorithm for constructing the

suffix tree of a given string in linear time. (II) Draw the implicit suffix tree and

show the list of rules used for each phase (i + 1) and each extension (j) to construct

the suffix tree for string “xabxababxba” by using the Ukkonen’s algorithm. (50%)

Sources: https://www.youtube.com/watch?v=1A6zrh7mfzg&ab\_channel=Keida

Three rules to building the suffix tree:

1) add character to the end of existing path

2) create a path if it doesn’t already and append the character to it

3) if the path already exists, do nothing

Ukkonen’s algorithm tricks:

1) skip/count and edge label compression

2) rule 3 extension is a show stopper (stop the current phase and start the next phase)

3) global end for all leaves

active points and rule three extension:

if you’re moving on the same edge then just increment your active length by 1

if you’re jumping the internal node then reset your active edge, active node, and active length variables

five variables to keep track:

remaining

end

active node

active edge

active length

Every substring is always a prefix of some suffix of the string

2. Suffix tree for large alphabet

When introducing the Ukkonen’s algorithm for suffix tree constructing, we as-

sume a constant size of the alphabet. If we assume the alphabet size |a| is compa-

rable to the length of the input string n, there is a trivial low bound O(n log n) for

applying Ukkonen’s algorithm. Describe a simple algorithm to achieve this lower

bound. (25%)

3. Peptide vaccine design

The activation of helper T-cells is essential to initiate a protective immune re-

sponse. To mimic pathogen invasion, biologists synthesize peptide vaccines, i.e.

small peptides of the essential proteins from a pathogen (bacterium or virus) that

can be recognized by the major histocompatibility complex (MHC) and presented

to the helper T-cells. A simple version of the peptide vaccine design problem can

be formulated as the shortest unique substring problem, which attempts to find the

shortest peptide in the proteins of the pathogen (called pathogen proteins) that are

not a part of any protein from the host (human) (called host proteins). (25%)

Pathogen host

abcd hthnh

egfgs

efghe