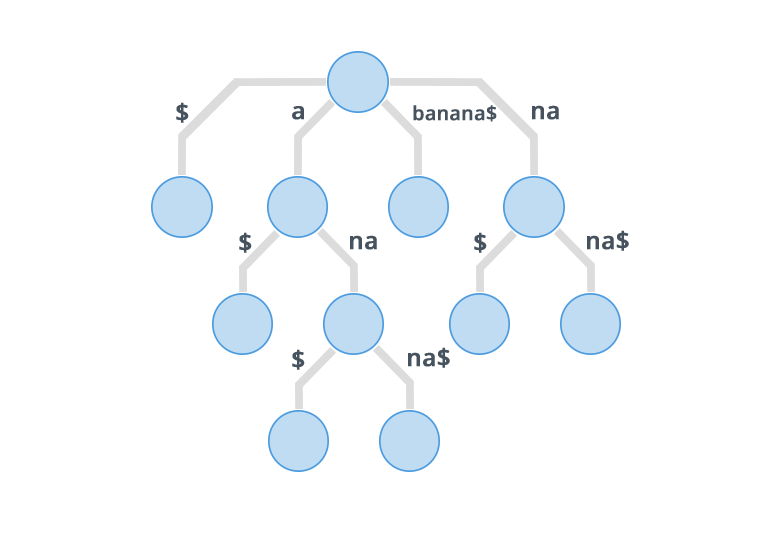
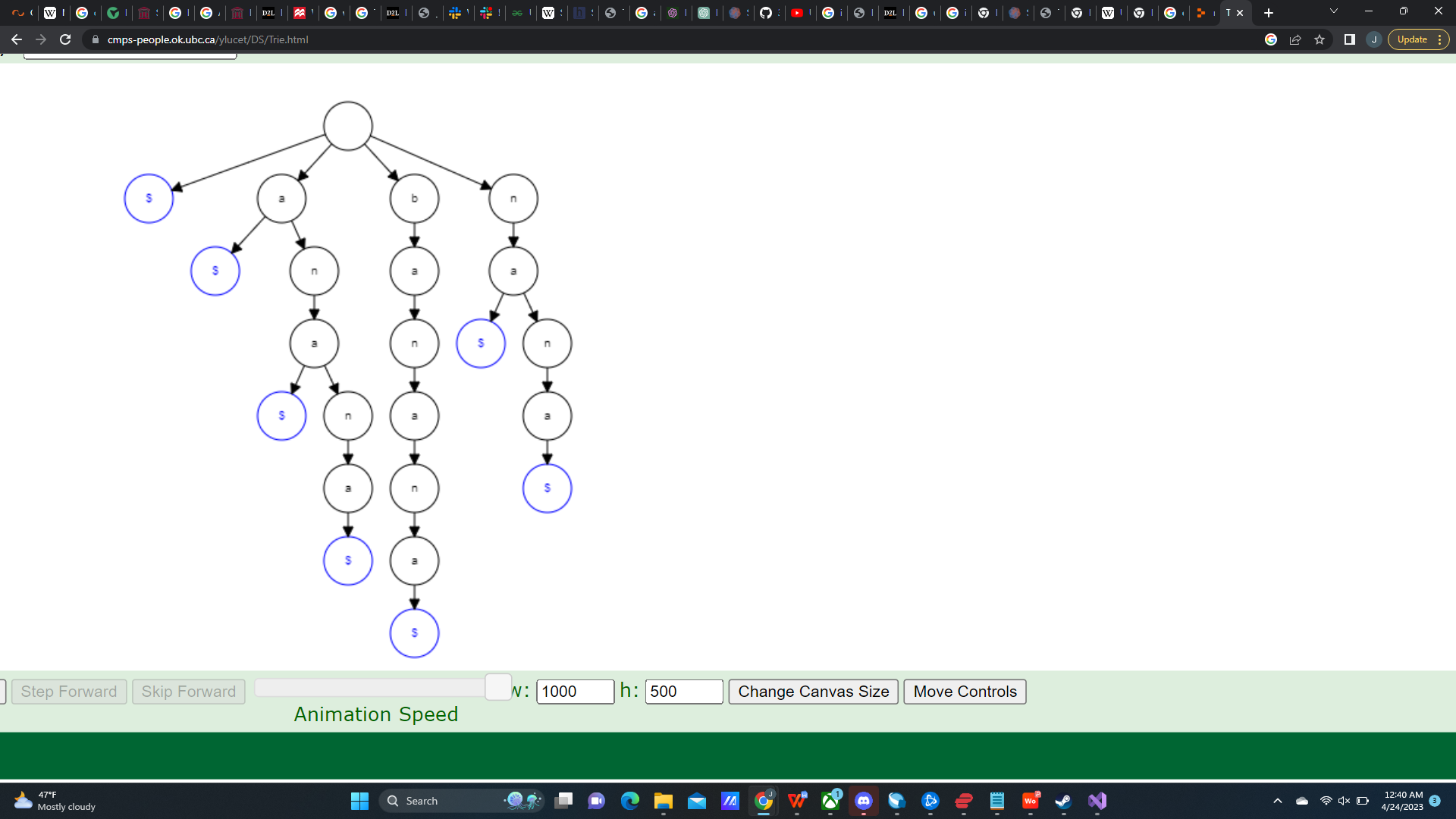
Suffix Tree Handouts

Is a compressed Trie tree of all the suffixes of a given string.This means this is a tree that joins all chain of nodes that only have one child.

for example lets pass in the word “**banana$**”

List of all suffix’s in the **banana$**

**banana$**

**anana$**

**nana$**

**ana$**

**na$**

**a$**

**$**

NB: to avoid the creation of an implicit suffix trees. Use a terminator character, in this case (**$),**which is a character that is not present anywhere else within the string.

An implicit suffix tree ignore repeated string patterns within a string which causes the tree to miss potential suffixes.

Steps for Inserting into a suffix tree (**Brute Force** ).

1) append unique character to the string you would like to insert

2) Perform normal Trie tree insert by inserting each suffix one at a time

Start recursive compression algorithm (starting at the root as the current node)

3) If the current node has a single child that is not a leaf node, merge the current node with its child, (concatenate characters) Update any relevant information

4) If the current node has multiple children, check if any of the children share the same edge label if so megre those children by concatenating label

5) of the current node is a leaf node ot multiple children with distinct edges, move on to the next child and repeat step 3 and 4

6)Once all children of the current node have been processed move onto the next child.

1. Repeat steps until completely processed

**Complexity**

Space complexity is **O(n)** which is an improvement from standard trie trees which are **O(n^2)**

The time complexity (Insertion) for the Brute force implementation of a suffix tree is **O(n^2)** but can be improved using the **Ukkonen Algorithm** which improves the complexity to **O(n).** This is because in the brute force algorithm inserts each different suffix individual while Ukkonen processes the string as its being inserted on its first go.

**Ukkonen Algorithm:**

proposed by Esko Ukkonen in 1995

The Ukkonen algorithm is categorized as an incremental and online algorithm. This is because this implementation allows for an incremental approach over a recursive approach.

**High Level Description**

Suffix extension is all about adding the next character into the suffix tree built so far. In extension j of phase i+j , algorithm finds the end of S[j..i] (which is already in the tree due to previous phase i) and then it extends S[j...i] to be sure the suffix S[j...i+1] is in the tree. There are three extension rules:

1) if the path from the root labeled S[j..i] ends at a leaf edge( i.e., S[i] is last character on leaf edge), the character S[i+1], then character S[i+1] is just added to the end of the label on that leaf edge.

2) if the path from the root labeled S[j..i] ends at a non - leaf edge (i.e, thee are more character after S[i[ on path) and next character is not S[i+1], then a new leaf edge with label S[i+1] and number j is created starting from character S[i+1]. A new internal node will also be created if S[1..i] ends inside (in between) a non-leaf leaf edge.

3) if the path from the root labeled S[j..i] ends at a non-leaf edge (i.e.,, there are more character after S[i] on path) and next character is S[i+1] (already in tree, do nothing.

One important point to note is that from a given node (root or internal), there will be one and only one edge starting from one character. There will not be more than one edge going out of any node starting with the same character