Eva Kuntz & Merritt Harvey

COMS311: Project 1

Analysis & Pseudocode

The following report contains pseudocode for the compute2k() method in each of the following classes: WarWithArray, WarWithBST, WarWithHash, and WarWithRollHash, in addition to the asymptotic run times of each of the algorithms.

1. WarWithArray

Here is the pseudocode for the compute2k() method:

For i in range [0….(n-1)] //O(n)

For j in range [0…(n-1)] //O(n)

Possible substring = string[i] + string[j];

For x in range [1…k] //O(k)

if possible.substring(x,x+k) not “valid”, exit loop

else, continue

the “valid” method is as follows:

isvalidSubstring(string s)

for I in range [0,….n-1] //O(n)

if s equals stringSet[i] //O(string length = k)

return true

Now, the outer loop in compute2k will run at most n times, the inner (j-indexed) loop will run at most n times. The x-indexed loop will run at most k times, and the ‘valid’ method has a loop that runs at most n times.

So, this method takes: n \* n \* k \* n \* k = n3k2 = O(k2n3) time in the worst-case.

1. WarWithBST

Here is the pseudocode for the compute2k() method:

For i in range [0….(n-1)] //O(n)

For j in range [0…(n-1)] //O(n)

Possible substring = string[i] + string[j];

For x in range [1…k] //O(k)

SearchBST for if possible.substring(x,x+k)

If not found, exit loop

Now, the outer i-indexed loop will run (at most) n times, and the inner j-indexed loop will take at most n times to complete. The x-indexed loop will run at most k times. The searchBST will take, at most, O(klogn) times, as we are comparing strings of length k at most logn times.

So, this method takes: n \* n \* k \* klogn = O(k2n2logn) time in the worst-case.

1. WarWithHash

Here is the pseudocode for the compute2k() method:

For i in range [0….(n-1)] //O(n)

For j in range [0…(n-1)] //O(n)

Possible substring = string[i] + string[j];

For x in range [1…k] //O(k)

Check if hashset contains possible.substring(x,x+k) //O(1)

If not found, exit loop

Now, the outer i-indexed loop will run (at most) n times, and the inner j-indexed loop will take at most n times to complete. The x-indexed loop will run at most k times, and accesses the hashset will take O(1).

So, this method takes: n \* n \* k \* 1 = O(kn2) time in the worst-case.

1. WarWithRollHash

Here is the pseudocode for the compute2k() method:

For i in range [0….(n-1)] //O(n)

For j in range [0…(n-1)] //O(n)

Possible substring = string[i] + string[j];

Hashcode = hashcode of possible.substring(1,1+k) //O(k)

For x in range [1…k] //O(k)

Hashcode = (hashcode – character(x-1))\*omega + character(k+x-1) //O(1)

Check if hashcode exists in table //O(1)

If not found, exit loop

Now, the outer i-indexed loop will run (at most) n times, and the inner j-indexed loop will take at most n times to complete. It takes O(k) to compute the hashcode of the initial substring, as there are k characters in the string. The x-indexed loop will run at most k times. Now, updating the hashcode is O(1) because we are manipulating integers (rolling hash), and checking if this key (long) is in the hashtable takes O(1) time.

So, this method takes: n \* n \* (k + k(1+1)) = O(kn2) time in the worst-case.