Prac12 Documentation

by Pavin Wu

Task 1

First, decompress the file by joining all the characters and their repeats in a single string variable *bwt*.

Then, to prepare to decode, create an array *occurrence* of size 256 to keep track of the occurrences of all the characters in *bwt* so far. Initialise this with 0's. Each entry corresponds to the character with the same code-point. (This implies this *occurrence* array is sorted based on the code-point value of each character.) Next, create *charNum* array to label each character in *bwt* with the occurrence it causes. So for all characters in *bwt*, update the character's occurrence in *occurrence*, then append the updated value to *charNum*.

Next, we create *rank* array. This *rank* array mimics to role of the first column of BW transform in that it stores the position in *bwt* where the first of each unique character occurs, and that the unique characters are sorted alphabetically (based on Unicode). However, notice that *occurrence* array is also sorted. This means we can create *rank* array simply by creating a dummy variable to represent a position in *bwt*, then add the number of occurrence of each unique character to this dummy variable to get the rank of the next character. Note that it is the 'next', not current character. This is because we store the rank as the 'first' occurrence, not the last.

Now that we have *charNum* and *rank* array, we simply do the decoding explained in the lecture, where the next row to go to is given by *rank[*code-point of character*]* + *charNum[*row of current character*]* – 1. Of course, this takes the advantage of the fact that relative occurrences amongst the same character are always the same for both the text *rank* array represents (1st col in BWT) or the *bwt* text (last col in BWT).

Space complexity: same as time complexity since every operation with O(1) time takes O(1) space.

Time complexity: decompressing text takes O(N), creating *charNum* (and *occurrence*) takes O(N), creating *rank* takes O(M) (since length of *occurrence*), and the decoding takes O(N). Thus, it takes total time of O(M+N).

Task 2a

Given $T[N] = 3*T[N/3] + aN$ and $T[1] = b$. Then

$T[N+1] = 3T[(N+1)/3] + a(N+1)$

$= 3(3T[(N+1)/9] + a((N+1)/3)) + a(N+1) \quad = 9T[(N+1)/9] + 2a(N+1) \quad = 3^2T[(N+1)/3^2] + 2a(N+1)$

$= 9(3T[(N+1)/27] + a((N+1)/9)) + 2a(N+1) = 27T[(N+1)/27] + 3a(N+1) = 3^3T[(N+1)/3^3] + 3a(N+1)$

$= \ldots = 3^kT[(N+1)/3^k] + ka(N+1)$ where $3^k = N+1$, or $k = \log_3[N+1]$

$= 3^kT[1] + ka(N+1) = 3^kb + ka(N+1)$

$= (N+1)b + \log_3[N+1]*a(N+1) = (N+1)(b+a\log_3[N+1])$

This is $O((N+1)\log_3[N+1]) = O(N\log[N])$

Task 2b

Base case: $T[1] = b$. This is self-evident.

Inductive step: Assume $T[N] = N(b+a\log_3 N)$, then

$T[3N] = 3N(b+a\log_3[3N]) = 3N(b+a\log_3[3]+a\log_3[N]) = 3N(b+a+a\log_3[N]) = 3N(b+a\log_3[N]) + 3Na$

$= 3T[N] + a(3N) = 3T[(3N)/3] + a(3N).$

Note that, from the given relation $T[N] = 3T[N/3]+aN$, the previous step leading up to step N is step N/3. Since we want to show the 'previous step' as step N (to prove with induction), we need to use step 3N as the 'next step' where step N leads up to.