Texas A&M University Kingsville
Department of EECS
CSEN 5303 Foundations of Computer Science
Project 3 Sorting

Mengxiang Jiang
Professor Habib Ammari

October 30, 2022

Contents

Chapter 1

Introduction

The problem given is to sort an array containing the characters 'A', 'M', and 'T', such that all the $T's$ appear first, followed by all the $A's$, and lastly all the $M's$. Special constraints are in place to make this sort harder:

-Constraint 1: Each letter ('A', 'M', or 'T') is evaluated **only once**.
-Constraint 2: The function $SWAP(TAM, i, j)$ is **used only** when it is **necessary**.
-Constraint 3: **No extra space** can be used by the algorithm $Sort\_TAM$. In other words, **only** the array TAMUK can be used to sort the 'A', 'M', or 'T'.
-Constraint 4: You **cannot count** the number of each letter 'A', 'M', or 'T'.

There is some ambiguity with contraints 2 and 3. One interpretation of constraint 2 is that only the minimum number of swaps needed to sort the array is allowed. However, I will show in chapter 2 that if this interpretation is followed, then constraint 3 will be violated, namely significantly more space is required to figure out the minimum amount of swaps. Therefore, my interpretation of constraint 2 is that swaps will only be used if the two letters being swapped are not the same. My interpretation of constraint 3 is that primitive data structures such as integers are allowed (such as indexing variables), just not something like bigger like arrays. The main algorithm used is a type of insertion sort, and the programming language used to implement is is Python.

Chapter 2

Design

First I will show the algorithm in pseudocode and then give a short example of how it works. Here is the pseudocode:

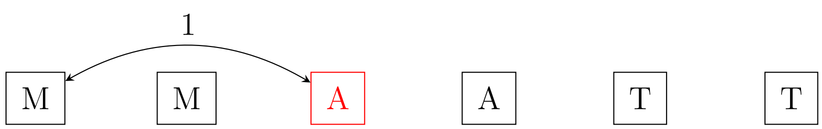
```
procedure Sort_TAM(TAMUK: arrayChar);
    var
        a, m, i: integer;
    begin
        a := -1;
        m := -1;
        for i:=0 to n-1 do
            if TAMUK[i] = 'T' then
                if a > -1 then
                    SWAP(TAMUK, a, i);
                    a := a + 1;
                else if m > -1 then
                    SWAP(TAMUK, m, i);
                    m := m + 1;
            if TAMUK[i] = 'A' then
                if m > -1 then
                    SWAP(TAMUK, m, i);
                    if a = -1 then
                        a := m;
                    m := m + 1;
                else if a = -1 then
                    a := i;
            if TAMUK[i] = 'M' then
                if m = -1 then
                    m := i;
    end;
```

Suppose the given character array, $TAMUK$, is given as below:

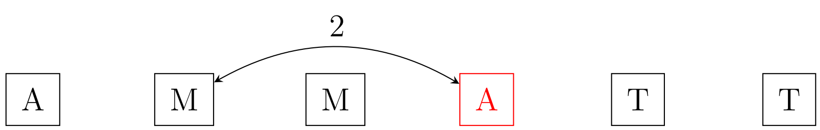
We initialize our variables: $a = -1$ and $m = -1$.

We evaluate $TAMUK[0]$: $a = -1$ and $m = 0$.

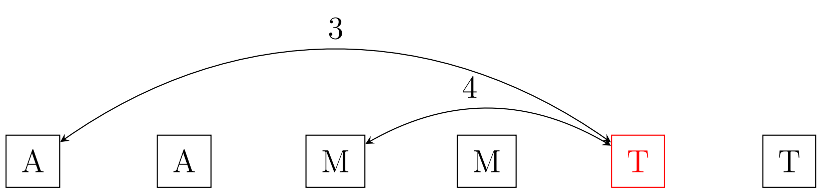
We evaluate $TAMUK[1]$: $a = -1$ and $m = 0$.



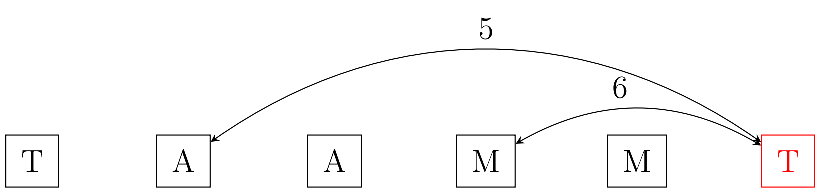
We evaluate $TAMUK[2]$: $a = 0$ and $m = 1$.



We evaluate $TAMUK[3]$: $a = 0$ and $m = 2$.

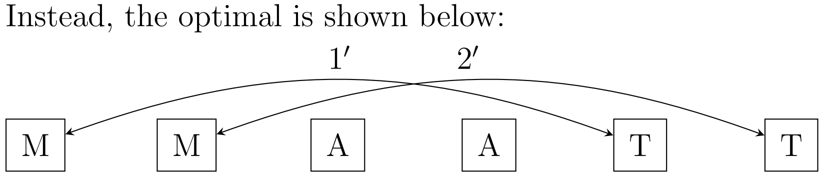


We evaluate $TAMUK[4]$: $a = 1$ and $m = 3$.



We evaluate $TAMUK[5]$: $a = 2$ and $m = 4$.

The final sorted array. Notice that there were 6 swap operations (indicated by the number on the directed edges). This is not the optimal/minimum number of swaps to sort the array. Instead, the optimal is shown below:



This optimal can be achieved by noticing that the swap operations are composible. Notice that the composition of swap 1, swap 3, and swap 4 is equivalent to swap $1'$, since 'A' starts and ends up at $TAMUK[2]$, 'M' starts at $TAMUK[0]$ and ends up at $TAMUK[4]$, and 'T' starts at $TAMUK[4]$ and ends up at $TAMUK[0]$. Similarly swap $2'$ is equivalent to the composition of swap 2, 5, and 6. Finding the minimal swap then becomes finding cycles in a graph. Unfortunately, we would then need to store the swaps as edges between vertices (which will be on the same order of magnitude as the array itself), violating constraint 3.

Chapter 3

Code

Chapter 4

Tests

Chapter 5

Lessons Learned