CS218 - Programming Assignment 2

Saksham Rathi
22B1003

1 Problem Description

This assignment involves ordering a range($[1 \dots n]$) for getting the optimal min peak memory requirement. For a particular order $(i_1, i_2, \dots, i_n)$, the peak memory requirement is defined as:

$$\max_{k \in \{1,2,\dots,n\}} \{m(i_k) + \sum_{1 \le p < k} m(i_p, i_k) + \sum_{k < q \le n} m(i_k, i_q) + \sum_{1 \le p < k} \sum_{k < q \le n} m(i_p, i_q)\} \tag{1}$$

We want to find a valid schedule that minimizes the peak memory requirement. We need to convert a naive $O(n! \times n^3)$ algorithm to a $O(2^n \times poly(n))$. The time complexity of my algorithm is $O(2^n \times n^2)$.

2 How my algorithm works?

My algorithm is based on Dynamic Programming. As, we can see in the equation 1, for a particular k, the order of the elements to the right of k does not matter. So, the $O(n!)$ naive algorithm has a lot of repititions. I have initialized the dynamic programming array to be of size (2^n) and all values are *INT_MAX* (because we are interested in the minimum over all orders). My dp array stores the optimal value of the above expression, when that particular subset is present in the left of the schedule which we wish to consider. Also, instead of iterating over the subsets directly, my algorithm iterates over the binary representation of these subsets. (For example 0101 means that out of the four elements a_0, a_1, a_2, a_3, only a_0 and a_2 are present.) After this we need to think about how we can move from a smaller subset to a larger subset. So, for a_0, a_1, a_2, we need to consider three subsets of size = 2 $\{(a_0, a_1), (a_1, a_2), (a_2, a_0)\}$. Let's say we have (a_0, a_1), so now we add a_2 at the end of this subset. After this we claim that the new "k" value will be either "k" of the previous smaller subset, or it will be at a_2 (newly inserted value). We can easily prove this because we are considering all three smaller subsets. So, even if the k value would have changed, then it will occur in one of the other subsets. So, dp value on the current subset will be maximum of the dp value of the previous subset and the value which we get after putting our k to the newly inserted element.

For optimizing our code further, I also have an auxiliary array of size 2^n, with all values initialized to zero. This array stores the following sum for a particular subset S:

$$\sum_{k \in S} \sum_{l \notin S} m(k, l)$$

This helps us in calculating our dp value of the current subset in $O(n)$ (we can access the auxiliary value of the previous subset and then update it).

So, overall, for every subset, we need to check $O(n)$ smaller subsets, and for moving from a smaller subset to a larger one, we again require $O(n)$ operations. We have 2^n subsets, so our overall code complexity is $O(2^n) \times n^2$.

(Also, for a paricular ordering, we are also supposed to check whether it is topologically valid or not.)