COMP3121-Ass3-Q5

z5302513, Kihwan Baek

July 2021

1. Assumption

We assume that V = $\{v_1, v_2, ..., v_n\}$ and opt(k,v_p,v_q) (1 $\leq$p$\leq$ n, 1 $\leq$q$\leq$ n) is the maximum total weight of path from a vertex v_p to a vertex v_q such that all intermediate vertices are among vertices $\{v_1, v_2, ...v_k\}$ ($1 \leq k \leq n$).

2. Base cases

The base case is opt(0,v_p,v_q) which is weight of path of length 1 starting with v_p and ending with v_q. We assume that if there is no edge between v_p and v_q, we set opt(1,v_p,v_q) as 'Inf' which means there is no path from v_p to v_q.

=> opt(0,v_p,v_q) = weight(v_p,v_q)

3. Subproblem and recursions

We use a slightly modified algorithm to get the maximum total weight of path. So, we compare opt(k,v_p,v_q) and opt(k-1,v_p,v_k) + opt(k-1,v_k,v_q). The difference between the former and the latter is whether or not it passes through the vertice v_k. We choose the larger of the former or the latter.

=> opt(k,v_p,v_q) = max$\{opt(k-1, v_p, v_q), opt(k-1, v_p, v_k)+opt(k-1, v_k, v_q)\}$...(1)

4. How to obtain the final solution and Time complexity

Firstly, we initialize n $\times$ n array with all weight(v_p, v_q) ($1 \leq p \leq n, 1 \leq q \leq n$). (Time Complexity = O($n^2$)) And then, we update the array using the recursion method. This process takes O(Kn^2) time because for all integer values ($0 \leq integer \leq$K), we need to check and update all array[i][j] (It takes O(K) $\times$ O(n^2) = O(Kn^2)).

At the same time, to get the exact path, we make new 2-dimentional array named next[][] and for all the edge between two vertices, set the next[][] value as the second vertice (i.e) if there is a edge between v_p and v_q set next[v_p][v_q] = v_q.

And also, for all vertices v, set the next[v][v] = v. When we use the recursion method, if $opt(k-1, v_p, v_q) < opt(k-1, v_p, v_k) + opt(k-1, v_k, v_q)$, we change next[i][j] to next[i][k] .

Hence, we can get the path as following next value of next[][] array (i.e) we need to get the weight of length 3 path from v_1 v_2, then next[1][2] = 2, next[2][1] = 1 and next[1][3] = 3. Then, the order of path is $1- > 2- > 1- > 3$ and this getting path process takes O(n^2+K) time.

Therefore, the total time complexity is$O(K)\times$ O(n^2) + O(n^2+K) = O(Kn^2).