Homecoming - Editorial

Bogdan Ciobanu, Andrei-Costin Constantinescu

9 July 2018

1 Minimum Cut / Greedy solutions

1.1 Subtask 1

The problem can be formulated as a max-flow min cut problem, being a particular instance of Project Selection. The network can be described as follows: the source is linked to the subjects $S_0, S_1, \dots S_{N-1}$ with capacities $A_0, A_1, \dots A_{N-1}$. The textbooks $T_0, T_1, \dots, T_{N-1}$ are linked to the sink with capacities $B_0, B_1, \dots B_{N-1}$. The i^{th} subject is then linked to its relevant textbooks with ∞ capacity. The answer is the sum of all possible profits $\sum\limits_{i=0}^{N-1} A_i$ minus the cost of the minimum cut in the network.

1.2 Subtask 2

The further optimize our previous solution we can reduce the number of edges. An easy idea which comes in handy is to consider a segment tree or a range minimum query-like sparse table over the subjects. This way we reduce the number of edges from $O(N \times K)$ to $O(N \log_2 N)$. A neat idea which applies here and yields only $O(N)$ edges is to consider the following easier problem: compute the result of an operation with no inverse over all subarrays of length K in $O(N)$ time. WLOG. suppose N is divisible by K, then a possible solution is along the lines of splitting the array in buckets of length K and computing the result of the operation in all buckets for every prefix and suffix. We can do something similar here, we can create the additional nodes $T_{P_0}, T_{P_1}, \dots, T_{P_{N-1}}$ and $T_{S_0}, T_{S_1}, \dots, T_{S_{N-1}}$. We will add the directed edges $T_{P_{K*i}} \to T_{P_{K*i+1}} \to \dots \to T_{P_{K*(i+1)-1}}$ and $T_{S_{K*(i+1)-1}} \leftarrow T_{P_{K*(i+1)-2}} \leftarrow \dots \leftarrow T_{S_{K*i}}$ with ∞ capacity. Now to link a subject to K consecutive textbooks we only need to add 2 edges.

1.3 Subtask 3

From now we can no longer rely on usual algorithms. We go back to the network from subtask 1 and we derived a different algorithm to compute the max flow. Consider a construction of $\sum_{i=0}^{N-1} B_i$ cells. Every subject may push the flow only in its interval $[\sum\limits_{j=0}^{i} B_i, \sum\limits_{j=0}^{i+K-1} B_i]$. We will keep buckets of pushed flow. For the first subjects we can push the flow as much we can to the left of the available interval. Obviously we will encounter difficulties only at the end, when the cyclic aspect comes in. In this case, we will push the buckets of flow from the prefix to the right, making sure to respect the interval ends. If we do it natively we get $O(K * N)$, because this $O(N)$ operation only for the last K.

1.4 Subtask 4

We can optimize the previous solution, for the last K subjects. When pushing on the prefix, we notice that every iteration in the algorithm will conceptually merge two intervals, so maintaining a stack of buckets of flow and recomputing its new border in order to enforce multiple constraints. This solution takes $O(N)$ amortized time.

2 Dynamic Programming solutions

2.1 Subtasks 1 and 2

We can first consider the solution which uses all subjects. If this is not the optimal one, then the one we're looking for has at least one subject which is not taken. We can fix this element and break the cycle around it, resulting in a solution of $O(N^3)$, or $O(N^2)$ if we use partial sums.

2.2 Subtask 3 and 4

We can optimize the previous solution if we fix a textbook out of the first K which is not taken or consider the solution in which all of them are taken, or even better, we can consider two cases: the one in which we take T_0 and the one in which we'll skip T_0. With proper care, both of these cases can be treated with (two possibly different) $O(N)$ dynamic programming solutions. The case in which we'll skip T_0 is treated with a pessimistic argument; the solution we obtain may not be optimal, but if it's not, we'll get the correct answer from the other case, because it will be optimal to take T_0.