

23. Consider the problem where the input is a collection of  $n$  train trips within Germany. For the  $i$ th trip  $T_i$  you are given the date  $d_i$  of that trip, and the non-discounted fare  $f_i$  for that trip. The German railway system sells a Bahncard for  $B$  Marks that entitles you to a 50% fare reduction on all train travel within Germany within  $L$  days of purchase. The problem is to determine when to buy a Bahncard to minimize the total cost of your travel.

Tree definition:

The tree is defined as being rooted at a “null” state where no decisions have been made. At each level,  $\ell$ , we may or may not purchase a Bahncard on  $d_\ell$ . If a card is purchased then we add  $B + \frac{f_\ell}{2}$  to the node’s value which is storing the total cost of all the trips. If the previously purchased Bahncard is still in effect add  $\frac{f_\ell}{2}$ . Otherwise we add  $f_\ell$ .

Pruning rules:

1. At each level  $\ell$ , there are multiple opportunities, 0 to  $\ell$ , for the “last card purchased”. Trips and cards purchased after level  $\ell$  are independent of the prior history, so we can prune all but the history with minimum cost. More succinctly: at every level  $\ell$ , prune all but the cheapest travel history with the same last-card-purchased.

Algorithm:

$$A[*, *] = \infty$$

$$A[0, 0] = 0. \text{ Initialize. } A[\text{level}, \text{last} - \text{purchase}] = \text{total} - \text{cost}.$$

For  $\ell = 0$  to  $n - 1$ :

For  $p = 0$  to  $\ell$ :

$$A[\ell + 1, \ell + 1] = \min(A[\ell + 1, \ell + 1], A[\ell, p] + B + \frac{f_{\ell+1}}{2}) \text{ Purchasing a new card and getting half off.}$$

If  $\ell - p < L$ : Previous card is still active get half off.

$$A[\ell + 1, p] = \min(A[\ell + 1, p], A[\ell, p] + \frac{f_{\ell+1}}{2})$$

Else: Pay full price.

$$A[\ell + 1, p] = \min(A[\ell + 1, p], A[\ell, p] + f_{\ell+1})$$

The solution will be found at  $\min(A[n, *])$

1. A square matrix  $M$  is lower triangular if each entry above the main diagonal is zero, that is, each entry  $M_{i,j}$ , with  $i < j$ , is equal to zero. Show that if there is an  $O(n^2)$  time algorithm for multiplying two  $n$  by  $n$  lower triangular matrices then there is an  $O(n^2)$  time algorithm for multiplying two arbitrary  $n$  by  $n$  matrices.

Given two  $n \times n$  matrices,  $M_1$  and  $M_2$ .

First, decompose  $M_1$  into a sum of lower- and upper- diagonal matrices  $L_1$  and  $U_1$  such that  $M_1 = L_1 + U_1$ . Decompose  $M_2$  in the same manner, so that  $M_2 = L_2 + U_2$ .

$$M_1 M_2 = (L_1 + U_1)(L_2 + U_2) = L_1 L_2 + L_1 U_2 + U_1 L_2 + U_1 U_2$$

Each component can be calculated in  $O(n^2)$  time, assuming there is an  $O(n^2)$  time algorithm for multiplying two  $n$  by  $n$  lower triangular matrices.

1. Calculating  $L_1 L_2$ : These are already both lower-triangular.

2. Calculating  $L_1 U_2$ :

$$\begin{bmatrix} 0 & 0 \\ 0 & L_1 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ U_2 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ L_1 U_2 & 0 \end{bmatrix}$$

3. Calculating  $U_1 L_2$ :

$$\begin{bmatrix} 0 & 0 \\ U_1 & 0 \end{bmatrix} \begin{bmatrix} L_2 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ U_1 L_2 & 0 \end{bmatrix}$$

4. Calculating  $U_1 U_2$ :

$$U_1 U_2 = (U_2^T U_1^T)^T$$