Dynamic programming - part 2

1. Discrete Knapsack problem

DP recurrence:

$$\underbrace{value[i][cap]}_{\text{max. profit obtained considering first i objects and using cap capacity}} = max\big(\underbrace{value[i\text{-}1][cap]}_{\text{object i is not added to the knapsack}}, \underbrace{value[i\text{-}1][cap\text{-}w[i]] + v[i]}_{\text{object i is added to the knapsack}}\big)$$

2. LIS (Longest increasing sequence problem)

DP recurrence (inefficient - O(n^2)):

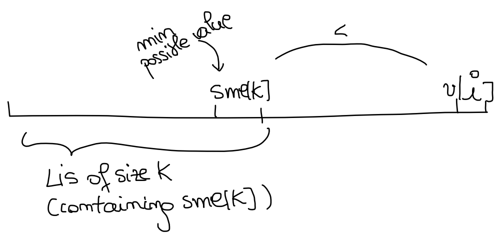
$$LIS[i] = \begin{cases} 1, & \text{if } \nexists\ 1 \le j \le i\text{-}1 \text{ with } v[j] < v[i] \\ max(LIS[j]) + 1, & \text{where } 1 \le j \le i\text{-}1 \text{ with } v[j] < v[i] \end{cases}$$

LIS that finishes with v[i]

DP recurrence (efficient - O(n*log(n))):

We define sml[len] = minimum element from v with which a longest increasing sequence of size len finishes

(1) $LIS[i] = \begin{cases} K+1, & \text{where K is the maximum index such that } sml[K] < v[i] \\ 1, & \text{if such K does not exist} \end{cases}$ ← K searched using binary search




(2) $sml[LIS[i]] = min(v[i], sml[LSi[i]])$

final element of an LSi of size LSi[i]