CS375 Notes

November 2, 2026

1 Recall: Candy Selection

We're given a list of candies to select from, where each candy has a joy j_i and a price p_i, and we want to maximize our joy subject to a budget of B.

$$\text{OPT}(i, b) = \text{“The best joy I can get from the first } i \text{ candies while spending at most } b\text{”}$$

$$\text{OPT}(i, b) = \max \begin{cases} j_i + \text{OPT}(i-1,\, b-p_i) & \text{if the } i\text{th candy is included,} \\ \text{OPT}(i-1,\, b) & \text{if the } i\text{th candy is not included.} \end{cases}$$

2 Longest Increasing Subsequence

Given a list of numbers $x_1, x_2, x_3, \ldots, x_n$, we want to find the longest increasing subsequence from them.

E.g.: 3 7 $\underline{1}$ $\underline{2}$ 6 $\underline{4}$ 9 $\underline{5}$ $\underline{8}$

Brute force: $O(n\,2^n)$ solution $\rightarrow$ try every subset.

Attempt 1. Say OPT(i) is the "longest increasing subsequence over just the first i numbers."

Think about the input $1, 4, 2, 3$: OPT tracks $1, 4$ as the LIS, while we could've had $1, 2, 3$.

Attempt 2. OPT(i) is the "longest increasing subsequence of the first i numbers that ends in the smallest number."

Think about the input $4, 5, 6, 1, 2, 3$: at the point we reach 3, OPT has $4, 5, 6$ with no information about $1, 2, 3$.

Attempt 3. $\text{OPT}_2(i)$: "The sequence of length 1 less than the LIS over the first i numbers that ends in the smallest number."

Using this on the previous example $4, 5, 6, 1, 2, 3$: after the first five numbers, OPT is $4, 5, 6$ and OPT_2 is $1, 2$ (the length-2 subsequence ending in the smallest number). When 3 arrives, it can extend OPT_2 to $1, 2, 3$, which has the same length as $4, 5, 6$ but ends in a smaller number, so OPT becomes $1, 2, 3$. This fixes the problem from Attempt 2.

But now OPT_2 has the same problem OPT had: to keep OPT_2 correct as new numbers arrive, we need the sequence that is 2 less than the LIS. (Here, $\text{OPT}_2 = 1, 2$ could only be found because we knew the length-1 sequence 1.) That sequence in turn needs the one that is 3 less, and so on down to length 1. Each of these is a similar problem to the previous OPT, so we need to track the smallest ending number for *every* length k. This leads to the final subproblem.

Final subproblem.

$\text{OPT}(i, k) =$ "The smallest last number of all length k subsequences over the first i numbers"

$$\text{OPT}(i, k) = \min \begin{cases} \text{OPT}(i-1,\, k) & \text{// ignore } x_i, \\ x_i & \text{// try to use } x_i \text{ — } \textbf{only if } x_i > \text{OPT}(i-1,\, k-1). \end{cases}$$

Base case: $\text{OPT}(i, k)$ with $i < k$: ∞.

Initial pseudocode.

```
1: Make an (n + 1) × (n + 1) array W
2: for i = 0, ..., n do
3:     for k = 0, ..., n do
4:         W[i, k] ← W[i − 1, k]
5:         if x_i > W[i − 1, k − 1] then
6:             W[i, k] ← min(W[i, k], x_i)
7:         end if
8:     end for
9: end for
```

Updated pseudocode.

```
1: Make an (n + 1) × (n + 1) array W
2: W[0, k] ← ∞ for k ≥ 1
3: W[i, 0] ← −∞ for i ≥ 0
4: for i = 1, ..., n do
5:     for k = 1, ..., i do
6:         W[i, k] ← W[i − 1, k]
7:         if x_i > W[i − 1, k − 1] then
8:             W[i, k] ← min(W[i, k], x_i)
9:         end if
10:    end for
11: end for
```