15-451/651 Algorithm Design & Analysis, Spring 2025

Homework #1 Solutions

1. (*, actual optimal static binary search tree)

   Recall the optimum binary search tree problem mentioned briefly at the end of Lecture 1: we want to arrange keys 1 to n in a binary tree, while minimizing

$$\sum_i \textsc{Depth}(i) \cdot w_i$$

   where $\textsc{Depth}(i)$ is the depth of i in the tree, and w_is are given weights. Given an $O(n^3)$ time algorithm for this problem.

   Note that this is not leaf-BST: internal nodes in the binary search tree should also have keys.

   References:

   - `https://en.wikipedia.org/wiki/Optimal_binary_search_tree`
   - `https://chatgpt.com/share/6966e15b-3b60-800c-9312-1ff633d5ad2e`, but up to the $n \le 300$ part...

   Solution: DP (interval DP). Let keys be $1, \dots, n$ with weights $w_1, \dots, w_n$.

   State. For $1 \le \ell \le r \le n$, define $DP[\ell][r]$ to be the minimum

$$\sum_{i=\ell}^{r} (\text{depth of key } i \text{ within the subtree}) \cdot w_i$$

   over all BSTs whose keys are exactly $\{\ell, \dots, r\}$, where the root of this subtree has depth 1. Define $DP[\ell][r] = 0$ for $\ell > r$ (empty tree).

   Prefix sums for weights. Let $P[0] = 0$ and $P[t] = \sum_{j=1}^{t} w_j$. Then for any interval $[\ell, r]$,

$$\text{SumW}(\ell, r) = \sum_{j=\ell}^{r} w_j = P[r] - P[\ell - 1].$$

   Transition. Choose the root to be $k \in \{\ell, \dots, r\}$. Then the left subtree uses keys $\{\ell, \dots, k-1\}$ and the right subtree uses keys $\{k+1, \dots, r\}$. When attaching these optimal subtrees under root k, every key in $[\ell, r]$ has its depth increased by 1 compared to treating subtrees as rooted at depth 0, which contributes exactly $\text{SumW}(\ell, r)$ in total weight. Thus:

$$DP[\ell][r] = \min_{k=\ell}^{r} \Big(DP[\ell][k-1] + DP[k+1][r] + \text{SumW}(\ell, r) \Big).$$

   Base cases. $DP[\ell][r] = 0$ if $\ell > r$. If $\ell = r$, then $DP[\ell][\ell] = \text{SumW}(\ell, \ell) = w_\ell$ (single root at depth 1).

   Output. The optimal BST objective value (with root depth 1) is $DP[1][n]$.

   Order of computation. Compute DP in increasing interval length $len = 1, 2, \dots, n$.

   Running time. There are $O(n^2)$ intervals (ℓ, r) and for each we try $O(n)$ roots k, each in $O(1)$ time using prefix sums. Total $O(n^3)$ time and $O(n^2)$ space.

2. (**, knapsack with weights from small ranges)

   Consider the value optimization version of knapsack without replacement: given n items with weights $w_1 \dots w_n$ and values $v_1 \dots v_n$, find a subset with weight at most W whose value is maximized. However, instead of W small, all weights are integers in a small range, from x to $x + k$.

   Give an algorithm for solving this problem that runs in $O(n^{10} k^{10})$ time or faster.

   Note:

- The 10s in exponents are to give some leeway: course staff are aware of $O(n^2k)$.

  Note (from post hw1 discussions): the intended solutions, e.g. the one below, take $O(n^3k)$ time. The reason we asked for $O(n^{10}k^{10})$ is to make it difficult to guess the solution structure / states from the intended bounds.

  The improved bounds come from using the fact that the 'extra' weights are in a size k range, and then incorporating tricks that are extensions of the random ordering optimization from Week 1 recitation. It is very much speculative, and likely omitted log factors.

  This remark was suppose to be speculative, and was not intended as guidance / hint for solutions. A more appropriate wording would have been "the fastest runtime that the course staff can imagine is around n^2k".
- When x is large, e.g. $x > n^{100}$, the $O(nW)$ bound we discussed in class can be much worse than this.
- the reference below gives $O((n/k)^k)$, which does not meet the runtime we ask here when k is large (e.g. $k = n/10$).

Reference: `https://atcoder.jp/contests/abc060/tasks/arc073_b`, with 3 replaced by k.

Solution: Let each weight be written as $w_i = x + \delta_i$ where $\delta_i \in \{0, 1, \ldots, k\}$. If we select exactly j items and the sum of their extras is $s := \sum \delta_i$, then the total weight is $jx + s$.

Define the DP:

$DP[i][j][s]$ = maximum total value achievable using items $1..i$, selecting exactly j items, with extra-sum exactly s.

Initialize

$$DP[0][0][0] = 0, \qquad DP[0][j][s] = -\infty \text{ for all other } (j, s).$$

Transition for item i (with value v_i and extra δ_i):

$$DP[i][j][s] = \max\Big(DP[i-1][j][s], \quad DP[i-1][j-1][s-\delta_i] + v_i\Big),$$

where the second term is only considered if $j \geq 1$ and $s \geq \delta_i$.

Finally, the answer is

$$\max\{DP[n][j][s] \mid 0 \leq j \leq n, \ 0 \leq s \leq jk, \ jx + s \leq W\}.$$

Running time. The indices satisfy $0 \leq j \leq i \leq n$ and $0 \leq s \leq jk \leq nk$, so there are $O(n^3k)$ states. Each transition is $O(1)$, giving total time $O(n^3k)$.

3. (counting counting knapsack)

   We have n items of positive integer weights, and a sum goal S.

   (a) (**, skip if you do part b) Give an algorithm that finds the number of subsets of these n items that sum to S in $O(nS)$ arithmetic operations.

   Solution: DP state. Let $dp_i[s]$ be the number of subsets of the first i items whose total weight is exactly s.

   Base cases. $dp_0[0] = 1$ and $dp_0[s] = 0$ for all $s > 0$.

   Transition. Let the i-th weight be a_i. For each $s \in \{0, 1, \ldots, S\}$,

   $$dp_i[s] = dp_{i-1}[s] + \mathbf{1}_{s \geq a_i} \cdot dp_{i-1}[s - a_i].$$

   (Exclude item i, or include it.)

   1D implementation. Maintain a single array $dp[0..S]$ initialized as $dp[0] = 1$, others 0. For each $i = 1..n$, update for $s = S, S-1, \ldots, a_i$:

   $$dp[s] \leftarrow dp[s] + dp[s - a_i].$$

Answer. Return $dp[S]$.

Running time. There are n iterations and each does $O(S)$ constant-time updates, so $O(nS)$ arithmetic operations.

(b) (***) Compute the **sum** of the answer of part (a) over all subsets of these n elements. That is, we want to sum over all subsets of these n elements, A the number of subsets of A that sum to S. Show that this is still computable in $O(nS)$ arithmetic operations.

hint: you can do this by adding 2 characters to a solution for part (a).

Solution: We want

$$\sum_{A\subseteq[n]}\left(\#\{B\subseteq A:\sum_{j\in B}a_j=S\}\right).$$

Equivalently, count pairs (A,B) such that $B\subseteq A\subseteq[n]$ and $\sum_{j\in B}a_j=S$.

For each item i, there are three possibilities:

- $i\in B$ (then automatically $i\in A$): adds weight a_i,
- $i\in A\setminus B$: adds weight 0,
- $i\notin A$: adds weight 0.

So "not in B" has $\underline{\text{two}}$ choices (in $A\setminus B$ or in neither), giving a factor 2.

DP state. Let $dp_i[s]$ be the number of valid pairs (A,B) using items $\{1,\dots,i\}$ with $\sum_{j\in B}a_j=s$.

Base cases. $dp_0[0]=1$, and $dp_0[s]=0$ for $s>0$.

Transition. For item i:

$$dp_i[s] = 2\cdot dp_{i-1}[s] + \mathbf{1}_{s\ge a_i}\cdot dp_{i-1}[s-a_i].$$

The $2\cdot dp_{i-1}[s]$ corresponds to the two choices where $i\notin B$, and the second term corresponds to choosing $i\in B$.

Answer. Return $dp[S]$.

Running time. Each item performs $O(S)$ constant-time arithmetic updates, for total $O(nS)$ arithmetic operations.

Note: the numbers that arise during these computations are enormous. Autograders keep this cost under control by asking for answer mod 998244353. For this course, we ask you to bound the number of arithmetic operations instead: you can assume all arithmetic, in particular, addition, and multiplication, take $O(1)$ time. Division/round are almost never needed for such problems: they are the point where arithmetic cost models can be severely abused.

Reference: `https://atcoder.jp/contests/abc169/tasks/abc169_f`