Dynamic Programming 1

Workshop in Competitive Programming – 234900
About Dynamic Programming

• Dynamic Programming ≈ Recursion + Re-use

• When to use?
  • Optimal substructure: an optimal solution can be constructed efficiently from optimal solutions of subproblems.
  • Overlapping subproblems.

• 3 steps for using DP:
  1. Define subproblem
  2. Find recurrence
  3. Solve base cases
Example: Number decompositions

- Problem: Given $n$, find the number of different ways to decompose $n$ as the sum of 1,3,4
- Example: for $n = 5$, the answer is 6:

$$5 = 1 + 1 + 1 + 1 + 1$$

$$= 1 + 1 + 3$$

$$= 1 + 3 + 1$$

$$= 3 + 1 + 1$$

$$= 1 + 4$$

$$= 4 + 1$$
Subproblems and Recurrence

• Define subproblems
  Let $D_n$ be the number of different ways to decompose $n$ as the sum of 1,3,4

• Find recurrence
  • For a possible solution $n = x_1 + x_2 + \cdots + x_m$
  • If $x_m = 1$, then $\sum_{i=1}^{m-1} x_i = n - 1$
    • The number of sums that end with $x_m = 1$ is $D_{n-1}$
    • Same goes for $x_m = 3$ and $x_m = 4$
  • Recurrence: $D_n = D_{n-1} + D_{n-3} + D_{n-4}$
Solve base cases

- \( D_0 = 1 \)
- \( D_n = 0 \) for all negative \( n \)
- Alternatively, set \( D_0 = D_1 = D_2 = 1 \), and \( D_3 = 2 \)

Implementation

\[
\begin{align*}
\text{for}(i &= 4; i <= n; i++) \\
    D[i] &= D[i-1] + D[i-3] + D[i-4];
\end{align*}
\]
Example: Knapsack

• **Goal:**
  
  • Given \( n \) elements with weights \( w_1, \ldots, w_n \) and values \( v_1, \ldots, v_n \), choose a subset of elements \( \{i_1, \ldots, i_k\} \) of maximum value, and weight below a certain threshold.

  \[
  \{i_1, \ldots, i_k\} = \arg\max\left\{ \sum v_{i_k} \mid \sum w_{i_k} \leq W \right\}
  \]
Knapsack – Solution using DP

• Define $F(i, w)$ - Maximal value we can obtain by selecting elements form 1, ..., $i$, such that the total weight is less or equal to $w$.

• Note that $F$ can be defined inductively:
  • Base case:
    \[ F(i, 0) = F(0, w) = 0 \]
  • Induction step (Assuming $w_i \leq W$):
    \[ F(i, w) = \max\{F(i - 1, w), F(i - 1, w - w_i) + v_i\} \]
    • Set $F(i, w) = F(i - 1, w)$ if $w_i > W$

• We are looking for the value $f(k, W)$
  • Can be computed using two nested loops in $O(k \cdot W)$
General Scheme

• **Goal**: Compute the value of a function $f$
  • The function can have one or many arguments

• **Assuming**:
  • $F(n_1, \ldots, n_k)$ can be defined *recursively*, without cyclic dependencies
  • Each argument has a finite number of possible value
  • The set of all possible parameter combinations is small enough to fit in memory

• If all assumptions hold, we can use **dynamic programming**, to efficiently compute the result.
General Scheme – cont.

• Therefore, for DP to work we will need to define:
  • **Stopping criteria**
    • e.g $F(n, 0) = 0, F(0, m) = 0$
  • **Target value**
    • e.g $F(N, M)$
  • **Recursive definition:**
    • e.g $F(n, m) = \max\{F(n - 1, m), F(n - 1, m - 1) + c(n)\}$
  • **Order of computation:**
    • e.g “For each $n$, loop over all values of $m$”
Subsets and Bitmasks

• Use an integer to represent subsets of a small set $S$:
  
  $\begin{align*}
  \text{ith digit} &= \begin{cases} 
  1, & i \notin S \\
  0, & i \in S 
  \end{cases} 
  \end{align*}$

  • Example: $19 = 010011_{(2)}$ represents a set $\{0, 1, 4\}$

• 2 sets $X$ and $Y$ in binary representation $x$ and $y$
  
  • Union: $x \mid y$
  
  • Intersection: $x \& y$
  
  • Symmetric difference: $x \^ \ y$
  
  • Singleton set $\{i\}$: $1 \ll i$
  
  • Membership test: $x \& (1 \ll i) \neq 0$