CMPSCI 311: Introduction to Algorithms

Lecture 12: Dynamic Programming 1

Akshay Krishnamurthy

University of Massachusetts

Last Compiled: March 19, 2018

Announcements

- Homework 2 graded, regrades open
- Homework 4 out Wednesday

Algorithm Design Techniques

- Greedy
- Divide and Conquer
- Dynamic Programming
- Network Flows

Dynamic Programming Schedule

- Today: Intro + Scheduling and Packing
- Thursday: Sequence Alignment + Biology problems
- 3/26: Graph problems
- 3/28: AI + Statistics problems

Divide and Conquer Recipe

- Devise recursive form for solution
- Implement recursion

Example Compute sum of leaf weights for each internal node in k-ary tree. (From practice exam)

- Recursive form $w(v) = \sum_{u \text{ child of } v} w(u)$.

Dynamic Programming Recipe

- Devise recursive form for solution
- Observe that recursive implementation involves redundant computation. (Often exponential time)
- Design iterative algorithm that solves all subproblems without redundancy.

Example (From HW1)

Problem. Given array A of length n, compute matrix B with $B[i,j] = A[i] + \ldots + A[j]$ for $i < j$.

for $i = 1, 2, \ldots, n$ **do**
  for $j = i+1, \ldots, n$ **do**
    Add up $A[i] + A[i+1] + ... + A[j]$. Store in $B[i,j]$.
  end for
end for

Running time: $\Theta(n^3)$.

Example (From HW1)

Problem. Compute B with $B[i,j] = A[i] + \ldots + A[j]$ for $i < j$.

$$B[i,j] = \begin{cases} B[i,j-1] + A[j] & \text{if } j > i \\ 0 & \text{if } j \leq i \end{cases}$$

for $i = 1, 2, \ldots, n$ **do**
  $B[i,i] = 0$
  for $j = i+1, \ldots, n$ **do**
    Add up $B[i,j-1] + A[j]$. Store in $B[i,j]$.
  end for
end for

Running time: $O(n^2)$

Weighted Interval Scheduling

- Television scheduling problem: Given n shows with start time s_i and finish time f_i, watch as many shows as possible, with no overlap.
- A Twist: Each show has a value v_i and want a set of shows S, with no overlap and maximum value $\sum_{i \in S} v_i$.
- Greedy?

Example

$$s = (0, 1, 4, 3, 7, 8)$$
$$f = (3, 5, 6, 9, 10, 11)$$
$$v = (2, 4, 4, 7, 2, 1)$$

Recursive Form

Order shows by finish time $f_1 \leq f_2, \ldots, \leq f_n$.

Compute $p(i) = \max\{j : f_j \leq s_i\}$ for all i.

- Suppose O is an optimal solution ($O = \mathrm{OPT}(n)$).
  - If $n \in O$, then $O = \mathrm{OPT}(p(n)) \cup \{n\}$.
  - If $n \notin O$ then $O = \mathrm{OPT}(n-1)$.
- Define $V = \mathrm{VAL}(n)$ to be the optimal value.
  - If $n \in O$, then $V = \mathrm{VAL}(p(n)) + v_n$.
  - If $n \notin O$, then $V = \mathrm{VAL}(n-1)$.

Recurrence $\mathrm{VAL}(n) = \max\{\mathrm{VAL}(p(n)) + v_n, \mathrm{VAL}(n-1)\}$.

Unrolling recurrence?

`Val`(j):
  If $j = 0$ return 0.
  Return $\max\{\texttt{Val}(p(j)) + v_j, \texttt{Val}(j-1)\}$.

- `Val(n)` can require 2^n calls in the worst case.
- Only $n+1$ values to compute $\Rightarrow$ redundancy!

Memoized approach

Idea. Save the output of recursive calls when you do them.

Array $M[0...n] = \texttt{null}$.
`M-Val(j)`:
  If $j = 0$ return 0.
  $M[j] \neq \texttt{null}$, return $M[j]$.
  $M[j] \leftarrow \max\{v_j + \texttt{M-Val}(p(j)), \texttt{M-Val}(j-1)\}$.
  Return $M[j]$.

Running time: $O(n)$.

Iterative approach

Idea. Work from $0 \rightarrow n$ computing array entries only once.

Array $M[0..n] = \texttt{null}$.
`I-All-Vals(n)`:
  $M[0] = 0$.
  for $j = 1, \ldots, n$ **do**
    $M[j] \leftarrow \max\{v_j + M[p(j)], M[j-1]\}$.
  end for

Running time: $O(n)$.

Finding the optimum set

- Suppose O is an optimal solution ($O = \mathrm{OPT}(n)$).
  - If $n \in O$, then $O = \mathrm{OPT}(p(n)) \cup \{n\}$.
  - If $n \notin O$ then $O = \mathrm{OPT}(n-1)$.

`Weighted-IS(n)`
  Sort by finish time f_j, compute $p(j)$.
  $M \leftarrow$ `I-All-Vals(n)`   # Compute M array
  $S \leftarrow \{\}, j = n$.
  while $j \neq 0$ **do**
    If $M[p(j)] + v_j \geq M[j-1]$, $S \leftarrow S \cup \{j\}$, $j \leftarrow p(j)$.
    Else $j \leftarrow j - 1$.
  end while
  Return S.

Weighted Interval Scheduling Takeaways

- Solution has recursive form.
- Can avoid unraveling the entire recursion.
- Dynamic Programming Table. The M array.
- Compute optimal value first, finding solution is easy after.