Dynamic Programming: I

Lecture 7

Akshar Varma

17th July, 2023

CS3000 Algorithms and Data

Misc Introductory Stuff

Fibonacci

Abstract framework

Weighted Interval Scheduling

Thinking About DP

Maximum Grid Path Sum

Knapsack

1. Misc Introductory Stuff

Agenda

1. Dynamic Programming (DP)
   - Compare and Contrast to Greedy and Divide-and-Conquer
2. Fibonacci Sequence
3. Abstract framework for DP
4. Weighted Interval Scheduling
5. Maximum Grid Path Sum
6. Knapsack

Dynamic Programming compared to ...

Greedy:

1. Build up solution incrementally.
2. Be short-sighted; short term and long term are aligned.
3. Only uses some local criterion.

Divide-and-conquer:

1. Divide up problem into *separate* subproblems.
2. Solve each subproblem recursively.
3. Combine subproblem solutions into overall solution.

Dynamic Programming:

1. Divide up problem into *overlapping* subproblems.
2. Solve each subproblem recursively.
3. Combine subproblem solutions into overall solution.
4. Compared to greedy, this manages to keep global goal in mind.

DP: History, Applications and Algorithms

- History “That is a made up name”

  Richard Bellman pioneered the paradigm in the 1950s, and made this name up, either

  - to avoid conflicts with the Secretary of Defense;
  - or to upstage Linear Programming.

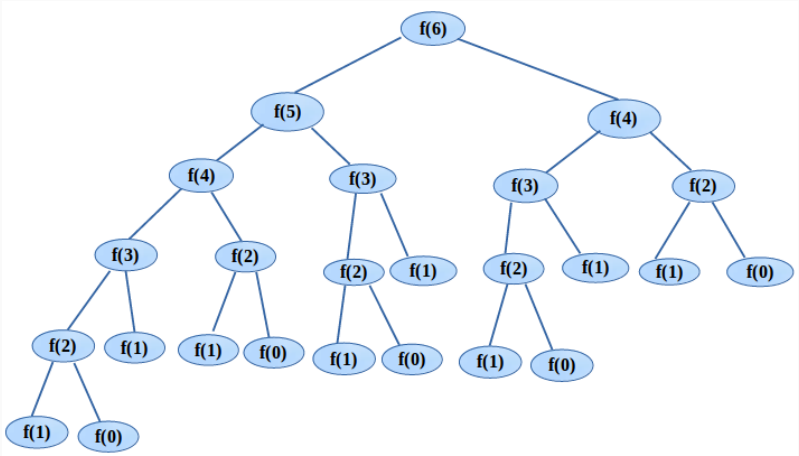
  In either case, the particular choice was made so that it sounded impressive. Something like “multistage decision making” describes the idea better.
- *Applications:* Throughout Computer Science, Operations Research, Control Theory, Bioinformatics, Economics, etc.
- *Algorithms:* Bellman-Ford for shortest path, Kadane (maximum subarray problem), Unix `diff` command, Viterbi Algorithm for HMMs, Cocke–Kasami–Younger (parsing CFGs), Knuth-Plass for word wrapping in TEX, Needleman–Wunsch sequence alignment (Bioinformatics), Duckworth–Lewis method (resolving interruptions in Cricket).

2. Fibonacci

This Fibonacci joke is as bad as the last two you heard combined.

Fibonacci

The Fibonacci sequence is defined recursively as: $F_n = F_{n-1} + F_{n-2}$



Recursion Tree for 6th term of the Fibonacci Sequence.

Those who cannot remember the past are condemned to repeat it.

– Dynamic Programming

DP to the rescue

- If we've already calculated a value, we should not forget it.
- Think about it as a table we need to fill in for later use.

n	F_n
1	1
2	1
3	2
4	3
5	5
6	8

- *Memoization*: "That is *not* a made up name."
  - Maintain a table of already solved subproblems.
  - Use them when we need those solutions again.
  - Name comes from memorandum/memo: *turning [the results of] a function into something to be remembered.*

3. Abstract framework

This Fibonacci joke is as bad as the last two you heard combined.

DP: The abstract idea

Optimal Substructure:

- Similar to Divide-and-Conquer.
- Optimally solving subproblems $\rightarrow$ optimal solution to overall problem.
- "Bellman Equation" of optimality. Example: $F_n = F_{n-1} + F_{n-2}$

Overlapping subproblems:

- DP would be the same as Divide-and-Conquer without this.
- Overlapping subproblems means you solve the same thing repeateadly.
- This is where memoization helps.

Solving using DP:

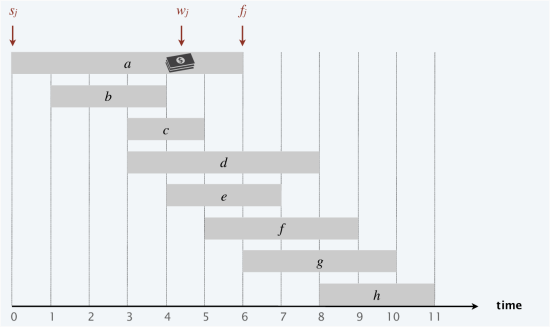
- Divide into *overlapping* subproblems. (Bellman Equation)
- Solve subproblems recursively (or bottom-up). (memoize/table filling)
- Combine subproblem solutions. (output a table entry)

4. Weighted Interval Scheduling

This Fibonacci joke is as bad as the last two you heard combined.

Weighted Interval Scheduling (WIS)

- Jobs start at s_i and finish at f_i.
- If two jobs overlap, they are incompatible.
- Each job also comes with an associated weight w_i.
- Goal: Find maximum weight subset of mutually compatible jobs.



Solving WIS using DP

- Rewrite so that we try small inputs and discover what information from the past we need.
- Now, if we solve for all jobs $1, \ldots, i-1$ can we solve until job i?
- Reformulated goal: Sort by f_i and then pick max jobs from $1, \ldots, n$.

- Define $OPT(i)$ as a the maximum weight solution using jobs 1 to i.
  If we *do not* pick job i, we can use $OPT(i-1)$.
  If we *do* pick job i, can't pick anything that conflicts with i.

- Find largest index $j < i$ that doesn't conflict with i and pick $OPT(j)$.
  Define this index value j to be p_i. Can find this via Binary Search.

- We now have the necessary ideas to find $OPT(n)$.

Bellman Optimality Equation for WIS

- We have everything to get the Bellman Optimality Equation.
- Writing it out formally helps understanding and also implementation.
- How to write a recursive equation for $OPT(i)$?
- First, the base case: $OPT(0) = 0$.
- Job i is either in the true optimal or it isn't.
  1. If i is not in the true optimal, then $OPT(i) = OPT(i-1)$.
  2. If i is in the true optimal, then $OPT(i) = w_i + OPT(p_i)$.
- Is i in the true optimal or not?
- When life gives DP options, try them *all* and then take the *best*.
- Bellman Optimality Equation:

$$OPT(i) = \begin{cases} 0 & i = 0 \\ \max\{OPT(i-1), \;\; w_i + OPT(p_i)\} & i > 0 \end{cases}$$

5. Thinking About DP

“Those who cannot remember the past are condemned to repeat it.”
– Dynamic Programming

How to think about DP

- You need to figure out the subproblem structure.
- You need to have the Bellman Optimality Equation.
- These tell you the table you'll fill.
- Sometimes it is easier to start with the table and work the rest out.
- Sometimes bottom-up may be easier.

6. Maximum Grid Path Sum

Maximum Grid Path Sum (Max GPS?)

- **Input:** An $n \times m$ grid of positive numbers.
- **Path:** Starts at the top left cell, moves right or down at each step and ends at bottom right cell.
- **Path sum:** The sum of elements in the path.
- **Goal:** Find maximum path sum among all paths.

Example Input:

1	5	1	9
2	7	8	2
8	2	6	4

Example output: 31.
R-D-R-D-R: 1+5+7+8+6+4 gives the maximum sum of 31.

Thinking about Max GPS

- Greedy that searches immediate neighbors fails.

1	1	1	100
9	1	1	1
9	9	9	1

- Maybe Brute Force? But how many paths exist in an $n \times m$ grid?
  - $= \binom{n-1+m-1}{m-1} \approx 2^{2n}$
- What is minimum sum you'll always get?
  - Top-left value + Bottom-right value.
- If grid size is 1×1, what's the solution? Top-left. Base case!
- If grid size is 2×2? Grid size 2×3? Grid size 3×2?
- Aha! Subproblems are $n - 1 \times m$ and $n \times m - 1$ (provided they exist).
- Bellman Optimality Equation:

$$S(n, m) = A[n, m] + \max\{S(n - 1, m), S(n, m - 1)\}$$

- Base case, Bellman Optimality Equation, Table. DP solution done!

Example run of DP for Max GPS

1	5	1	9
2	7	8	2
8	2	6	4

Memoization: The table we fill

1	6	7	16
3	13	21	23
11	15	27	31

7. Knapsack

The Knapsack Problem

- **Input:**
  - n items, each with a value $v_i > 0$ and a weight $w_i > 0$. $v_i, w_i \in \mathbb{N}$.
  - Knapsack that has a capacity of W.
- **Goal:** Pack knapsack to maximum value possible.
- **Example:** If $W = 11$ and

i	v_i	w_i
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

- $\{1, 2, 5\}$ has value 35, weight 10. (pick largest value greedily)
- $\{3, 4\}$ has value 40, weight 11. (be smarter in picking)

Knapsack via DP

- There is a lot of similarity to WIS. Pick or not pick item.
- But weight limit causes conflicts, not start/end times.
- We need to consider which item gets picked *and* how much it weighs.
- Define: $OPT(i, w)$ as the max profit weight of items $1, \ldots, i$ with weight limit w. So Goal is $OPT(n, W)$.
- Filling in $OPT(i, w)$ follows a similar argument to WIS.
  1. Case 1: True optimal does not pick i: $OPT(i, w) = OPT(i-1, w)$
  2. Case 2: True optimal does pick i: $OPT(i, w) = v_i + OPT(i-1, w-w_i)$
     Get v_i; incur w_i. So we can pick best from $i-1$ using $w - w_i$ budget.
- Bellman Optimality Equation

$$OPT(i, w) = \begin{cases} 0 & i = 0 \\ OPT(i-1, w) & w_i > w \\ \max\{OPT(i-1, w), \;\; v_i + OPT(i-1, w-w_i)\} & else \end{cases}$$

Summary

- General structure of Dynamic Programming
  - Break into subproblems; will overlap, unlike Divide-and-Conquer.
  - Make sure you have a Bellman Optimality Equation.
  - Memoize/Fill table.
  - Read out correct table entry.
- Figuring out the right subproblem structure to exploit ~~is an art~~ takes practice.
- Make sure the equation is correct before you start implementing.