Dynamic Programming: LIS, Partitioning Books, and Edit Distance

Admin

- Student solutions updated
- Thursday 6:15-8:15 and 8-10 TA office hours moved to Saturday 4-6PM and 6-8PM
  - Check the schedule

Study Buddy System

- Less socializing this semester!
- Email me and I'll set up groups of 3-4 students
  - sam@cs.williams.edu
- Up to you what you do in these groups
- Ideally something like:
  - Ask each other questions
  - Work through assignment questions or other questions in the textbook
- Reading proofs to each other **will** help you in this class.

Longest Increasing Subsequence

- Given a sequence of integers as an array $A[1,\ldots n]$, find the longest subsequence whose elements are in increasing order
- Find the longest possible sequence of indices $1 \le i_1 < i_2 < \ldots < i_\ell \le n$ such that $A[i_k] < A[i_{k+1}]$
- E.g., $A = [3\ ,\ 8\ ,\ 4\ ,\ 5\ ,\ 9\ ,\ 2]$
- The longest increasing subsequence of A is $1,\ 4,\ 5,\ 9$
- Length of the longest increasing subsequence is 4
- To simplify, we will only compute **length of the LIS**

LIS: Recursive Subproblem

- The most important part of a dynamic program: **subproblems**
- **Subproblem.** $L[i]$ denote the length of the **longest increasing subsequence** that ends in $A[i]$
- **Our goal** (in terms of the subproblem): $\max_{1 \le i \le n} L[i]$
- **Base case.** $L[1] = 1$
- How do we go from one subproblem to the next, that is, how do we compute $L[i]$ assuming I know the values of $L[1], \ \ldots, \ L[i-1]$

1 **2** 10 **3** 7 6 **4** **8** **11**

Recurrence

- Let's say we know the length of the longest subsequence ending at $A[1], A[2], \ldots A[i-1]$
- What is the longest subsequence ending at $A[i]$?
- Must be:
  - The longest subsequence ending at some $A[k]$
  - With $A[k] < A[i]$
- OK, let's try all k to get the answer

Towards a Recurrence

- Let us take an example $A = [5,\ 2,\ 8,\ 6,\ 3,\ 6,\ 9,\ 7]$
- $L[1] = 1$ (just the sequence 5)
- What is $L[2]$?
    - Since $A[2] < A[1]$ it does not extend prev LIS, so it starts a new one: just 2
    - $L(2) = 1$
- What about $L(3)$?
    - Since $A(3) > A(1)$ and $A(3) > A(2)$ is extends both subsequences, maximum length is 2
    - $L(3) = 2$

Towards a Recurrence

- Let us take an example $A = [5,\ 2,\ 8,\ 6,\ 3,\ 6,\ 9,\ 7]$
- So far $L(1,3) = [1,1,2]$
- What about $L(4)$?
  - Either it extends a prev LIS ending at $A[1], A[2], A[3]$ (and we take maximum) + 1
  - Or it doesn't extend any of them, and $L(4) = 1$
  - Do we need to remember the subsequences to check this?
  - No we just see if $A(4) > A(i)$ for $i = 1,2,3$
  - Or it doesn't extend any of them, and $L(4) = 1$

LIS: Recursive Subproblem

- $L(j) = 1 + \max\{L(i) \mid i < j \text{ and } A[i] < A[j]\}$
    - Assuming $\max \varnothing = 0$

Recursion to Dynamic Program

- If we used recursion (without memoization) we'll be inefficient—we'll do a lot of repeated work
- Once you have your recurrence, the remaining pieces of the dynamic programming algorithm are
  - **Evaluation order.** In what order should I evaluate my subproblems so that everything I need to evaluate a new subproblem?
    - For LIS we just left-to-right on array indices
  - **Memoization structure.** Need a table (array or multi-dimensional array) to store computed values
    - For LIS, we just need a one dimensional array
    - For others, we may need a table (two-dimensional array)

Dynamic Programming Practice

- Suppose we have to scan through a shelf of books, and this task can be split between k workers
- We do not want to reorder/rearrange the books, so instead we divide the shelf into k regions
- Each worker is assigned one of the regions
- What is the fairest way to divide the shelf up?

DP: Dividing Work

- Suppose we have to scan through a shelf of books, and this task can be split between k workers
- We do not want to reorder/rearrange the books, so instead we divide the shelf into k regions
- Each worker is assigned one of the regions
- What is the fairest way to divide the shelf up?
- If the books are equal length, we can just give each worker the same number of books
- What if books are not equal size?
    - How can we find the fairest partition of work?

The Linear Partition Problem

- **Input.** A input arrangement S of nonnegative integers $\{s_1, \ldots, s_n\}$ and an integer k
- **Problem.** Partition S into k ranges such that the maximum sum over all the ranges is minimized
- **Example.**
  - Consider the following arrangement

    100 200 300 400 500 600 700 800 900

  - Suppose $k = 3$, where should we partition to minimize the maximum sum over all ranges?

    100 200 300 400 500 | 600 700 | 800 900

Optimal Substructure

- Notice that the kth partition starts after we place the $(k-1)$st "divider"
- Let us try to construct an optimal solution. Where can we place the *last* divider?
  - Between some elements, suppose between ith and $(i+1)$st element where $1 \leq i \leq n-1$
  - What is the cost of placing the last divider here? Max of:
    - Cost of the last partition $\sum_{j=i+1}^{n} s_j$
    - Cost of the optimal way to partition the elements to the "left" — this is a smaller version of the same problem!
- **Question:** Can you come up with the **subproblem** for the dynamic program?

Dividing Work: DP Algorithm

- **Subproblem**. $M[i, j]$ be the minimum cost over all partitions of first i books into j partitions, $1 \leq i \leq n$, $1 \leq j \leq k$
- **Base cases**.
  - $M[1, j] = s_1$ for all $1 \leq j \leq k$
  - $M[i, 1] = \sum_{t=1}^{i} s_t$ for all $1 \leq i \leq n$
- **Recurrence**.
  - Dictates how we go from one subproblem to the next
  - Now we have a two dimensional table so we also need to think about which order to go in (what the dependencies are…)

Dividing Work: DP Algorithm

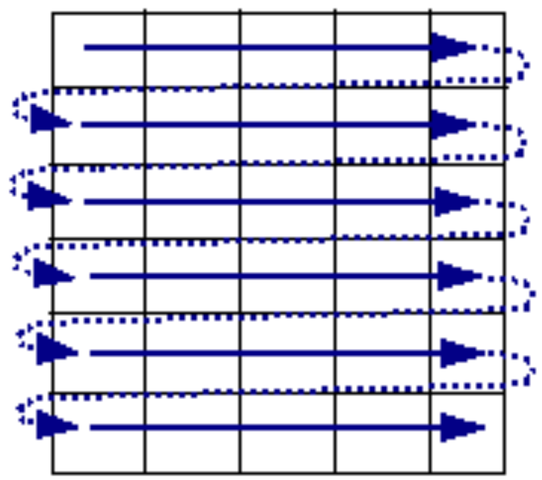
- **Subproblem**. $M[i, j]$ be the minimum cost over all partitions of first i books into j partitions, $1 \leq i \leq n$, $1 \leq j \leq k$
- **Base cases**.
  - $M[1, j] = s_1$ for all $1 \leq j \leq k$
  - $M[i, 1] = \sum_{t=1}^{i} s_t$ for all $1 \leq i \leq n$
- **Recurrence**. $M[i, j] = \min_{1 \leq i' \leq i} \max\{M(i', j-1), \sum_{t=i'+1}^{i} s_t\}$
- **Final solution**. $M[n, k]$
- **Memoization structure**. Two-dimensional array.
- **Evaluation order**. ?

Evaluation Order

- What do we need filled in so that we can fill in $M[i, j]$?
- For all $i' < i$, need $M[i', j - 1]$
- Plan: fill in all $M[i,1]$, then all $M[i,2]$ (in increasing order of i), then all $M[i,3]$, and so on
- Let's draw out M where each value of j is a row of M

Dividing Work: Final Pieces

- Evaluation order.
    - To fill out one cell, we need to take min over the values to the left in the previous row
    - Thus, we fill out rows one-by-one
    - Called row major order
- Running time?
    - Size of table: $O(k \cdot n)$
    - How long to compute a single cell?
        - Depends on n other cells
    - $O(n^2 \cdot k)$ time



Row-major order

Running Time

- Running time
    - Size of table: $O(k \cdot n)$
    - How long to compute a single cell?
        - Depends on n other cells
    - $O(n^2 \cdot k)$ time
- Is this a polynomial running time?
- How big can k get?
    - At most n non-empty partitions of n elements
    - $O(n^3)$ algorithm in the worst case

Recipe for a Dynamic Program

- **Formulate the right subproblem.** The subproblem must have an optimal substructure
- **Formulate the recurrence.** Identify how the result of the smaller subproblems can lead to that of a larger subproblem
- **State the base case(s)**. The subproblem thats so small we know the answer to it!
- **State the final answer.** (In terms of the subproblem)
- **Choose a memoization data structure.** Where are you going to store already computed results? (Usually a table)
- **Identify evaluation order.** Identify the dependencies: which subproblems depend on which ones? Using these dependencies, identify an evaluation order
- **Analyze space and running time.** As always!

Acknowledgments

- Some of the material in these slides are taken from
  - Kleinberg Tardos Slides by Kevin Wayne (https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/04GreedyAlgorithmsI.pdf)
  - Jeff Erickson's Algorithms Book (http://jeffe.cs.illinois.edu/teaching/algorithms/book/Algorithms-JeffE.pdf)