Program Structure and Algorithms

Sid Nath

Lecture 14

Agenda

- Administrative
  - Please check your scores and report any discrepancies by April 20, 2025 11:59pm!

- Lecture
  - Final Review

- Quiz

Final Exam Details

- Closed book/notes, only two A4-sized cheatsheets and calculator allowed!
- ~3 hours
- Q1: Short answers
- Q2: Dijkstra execution
- Q3: Greedy execution
- Q4: Greedy algorithm development
- Q6: DP algorithm development
- Q7 (Bonus): DP algorithm development

Be concise, to the point, answer what is asked

Greedy Review

- Greedy choice

Problem	Type: min/max	Greedy choice
Dijkstra's	Min path length	Shortest distance from every visited edge
Huffman's coding	Min symbols	Smallest freq character first
Interval scheduling / activity selection	Max #activities for a resource	Earliest finish times
Interval partitioning	Min #classrooms / resources	Earliest start times
Minimum spanning trees	Min cost tree	Lightest edge first
Coin change (US currency)	Min #coins	Highest denomination first
Fractional knapsack	Max value in knapsack	Highest value/weight ratio first

DP: Chain of Thought

- What are my subproblems?
- What are the decisions to solve each subproblem?
- Recursive formulation
  - Base case
- How many subproblems? What is the running time per subproblem?
- What is the overall running time?

DP: Common Subproblems

Finding the right subproblem takes creativity and experimentation. Some standard choices below

- Template #1: Input is $x_1, x_2, \dots, x_n, (or, x[1:n])$ and a subproblem is $x_1, x_2, \dots, x_i$ (or, $x[1:i])$
  - Number of subproblems is linear

- Template #2: Input is $x_1, x_2, \dots, x_n$, and a subproblem is $x_i, x_{i+1}, \dots, x_j$ (or, $x[i:j])$
  - #subproblems is quadratic $\mathrm{O}(n^2)$

- Template #3: Inputs are $x_1, x_2, \dots, x_n$, and $y_1, y_2, \dots, y_m$; a subproblem is $x_1, x_2, \dots, x_i$ and $y_1, y_2, \dots, y_j$
  - #subproblems is quadratric $\mathrm{O}(mn)$

DP: Useful Tricks for Template #1

- Template #1: Input is $x_1, x_2, \ldots, x_n$, ($or, x[1{:}\,n]$) and a subproblem is $x_1, x_2, \ldots, x_i$ (or, $x[1{:}\,i]$)
  - Number of subproblems is linear
  - Trick in the knapsack problem is to define subproblems based on max value for a smaller weight constraint
  - Trick in the share trading problem is to define subproblems based on max profit achieved on day of selling
  - Trick in the weighted interval scheduling problem is to preprocess ordering of dependent inputs and then define subproblems based on max weight subset on mutually compatible previous jobs

DP Problem #1

You are given a string s and a pattern p. Both s *and* p contain lowercase English letters, but p contains two extra characters "." and "*".

- "." matches any single character
- "*" matches zero or more occurrences of the element just before it

Your task is to check if p can match the entire string s. Devise an efficient algorithm for this task?

Examples

s = "abb", p = "a.b" or "ab*" → True

s = "aab", p = "ab." or "ab*" → False

s = "bad", p = "a*b.*d" ???

RegEx Matching using DP

This is not an optimization problem, but we can formulate it as DP

Consider the $s[1:i] = s_i$ and $p[1:j] = p_j$ characters

Substructure intuition

- $p[j] = "*"$
  - Pattern without "*" (i.e., zero occurrences), look at $p[j-2]$ with $s[i]$
  - Pattern with "*", valid only when either $\mathrm{s[i]} = p[j-1]$ or $\mathrm{p}[j-1] == "."$
- $p[j] = "."$
  - Continue with $s[i-1]$ and $p[j-1]$
- $s[i] = p[j]$, continue with $s[i-1]$ and $p[j-1]$
- $s[i] \mathrel{!}= p[j]$, return False

RegEx Matching using DP

Subproblem: $rem[i,j]$ = True if $s[1:i]$ can be constructed from $p[1:j]$, False otherwise

Decisions: (from previous slide)

$$rem[i,j] = \begin{cases} rem[i,j-2]\ if\ don't\ use\ p[j] == "*" \\ OR \\ rem[i-1,j]\ if\ use\ p[j] == "*"\ and \\ conditions\ on\ p[j-1]\ apply \\ rem[i-1,j-1]\ \ if\ s[i] == p[j]\ OR\ p[j] == "." \\ False, otherwise \end{cases}$$

Base case:

$$rem[0,0] = True, rem[i,0] = False$$

Running time: $O(mn) \times O(1) = O(mn)$

Regex Matching using DP

s = “bad”, p = “a*b.*d” ???

DP Problem #2

Given n coins $(C_1, C_2, \ldots, C_n)$ and their respective probabilities $(p_1, p_2, \ldots, p_n)$ of getting a head in a random toss. Some of the coins may be biased, that is, $p_i \neq 0.5$. You are also given a positive integer k.

Suppose you toss coins in order $(C_1, C_2, \ldots, C_n)$, what is probability of obtaining exactly k heads in n tosses? Devise a DP algorithm to solve this problem.

Small Example

Let n = 3, probabilities {1/3, ½, ¾} and k = 2.

How to get exactly 2 heads in 3 tosses?

HHT, HTH, THH

If p_i is probability of head, then probability of tail is $(1 - p_i)$

HHT = 1/3 * ½ * (1-3/4) = 1/24

HTH = 1/3 * (1-1/2) * ¾ = 3/24

THH = (1-1/3) * ½ * ¾ = 6/24

Total: 10/24 = 5/12

Brute Force Algorithm?

Enumerate all possible ways of obtaining exactly k heads in n tosses = C(n, k) ~ $O(2^n)$

DP Approach?

- Subproblems
  - We need to track two things -- # tosses and #heads – so need two variables
  - P(i, j) = prob of obtaining exactly j heads in the first i tosses of $(C_1, C_2, \ldots, C_i)$
- Decisions?
  - If ith toss is a H, then we need exactly (j-1) heads in prev (i-1) tosses; $P(i-1, j-1) \cdot p_i$
  - If ith toss is a T, then we need exactly j heads in prev (i-1) tosses; $P(i-1, j) \cdot (1-p_i)$
- Recursion
  - $P(i, j) = P(i-1, j-1) \cdot p_i + P(i-1, j) \cdot (1-p_i)$ if j >= 1

    $P(i-1, j) \cdot (1-p_i)$ if j = 0

DP Approach?

- Recursion
  - $P(i,j) = P(i-1,j-1) \cdot p_i + P(i-1,j) \cdot (1-p_i)$ if j >= 1

    $P(i-1,j) \cdot (1-p_i)$ if j = 0
- Base cases
  - P(0,j) = 0 for j <= i else 1; P(0, 0) = 1
  - P(1, j) = p_1 for j = 1 else (1- p_1)
  - P(i, j) = 0 j > i
- Running time
  - #subproblems: nk
  - Running time per subproblem: O(1)
  - Total running time: O(nk)

Lecture 14 summary

- Final Review

- All the best!!!