CS 312: Algorithm Analysis

Lecture #22: Intro. to Dynamic Programming

Slides by: Eric Ringger, with contributions from Mike Jones, Eric Mercer, Sean Warnick

This work is licensed under a Creative Commons Attribution-Share Alike 3.0 Unported License.

Objectives

- Motivate Dynamic Programming (DP)
- Understand the tabular approach to DP
- Use DP to solve an example problem

Families of Algorithms

- Introductory
  - Importance of algorithm analysis, seen through RSA
- Elegant Design Principles
  - Divide-and-Conquer
  - Graph Exploration
  - Greedy
- Drawback: only usable on very specific types of problems
- Need: “Sledgehammers of the algorithms craft”
  - Dynamic Programming
  - Linear Programming

Divide & Conquer

Dynamic Programming

Dynamic Programming

Divide a problem into smaller sub-problems at the bottom.

Build a table of results as you go.
**Dynamic Programming**

Avoid recomputing intermediate results; Consult the table

---

**Dynamic Programming**

Avoid recomputing intermediate results; Consult the table

---

**Dynamic Programming**

---

**Dynamic Programming**

---

**Dynamic Programming**

---

**Dynamic Programming**

---

**Dynamic Programming**

---

**Dynamic Programming**

---

**Dynamic Programming**

---

**DP Key Idea #1**

Not every sub-problem is new.

Save time: retain prior results.

---

**DP Key Idea #2**

- Devise a minimal description (address) for any problem instance and sub-problem
- Use this description as key to a table

---

**Dynamic Programming**

- Identify sub-problems
- Define the minimal description for each
  - Perspective #1: Divide & Conquer with Memory Table
    - Solve the sub-problems one by one, smallest first, possibly by recursion
    - Store the solutions to sub-problems in a table and reuse the solutions to solve larger sub-problems
    - Until the top (original) problem instance is solved
  - Perspective #2: DAG
    - Nodes are sub-problems; edges are dependencies between the sub-problems
    - Linearize!
    - Solve the sub-problems one by one in the linearized order
Example: Binomial Coefficients

How many ways are there to choose $k$ items from a set of $n$ items?

$$C(n, k) = \binom{n}{k} = \frac{n!}{k!(n-k)!}$$

$$= \begin{cases} 1 & \text{if } k = 0 \text{ or } k = n \\ \frac{(n-1)!}{(k-1)!(n-k)!} + \frac{(n-1)!}{k!(n-k)!} & \text{if } 0 < k < n \\ 0 & \text{otherwise} \end{cases}$$

Example: $C(5, 3)$

From DAG to Table

- What is the minimal description?
- Can you embed this DAG in a table?
- How many dimensions would you need?
- How would you index the sub-problems?

From Figure

- $C(n, k) = \binom{n}{k}$
- $\Omega(C(n, k))$

Pascal's Triangle

Blaise Pascal (1623-1662)
- Second person to invent the calculator
- Religious philosopher
- Mathematician and physicist
DP: From Problem to Table to Algorithm

- Start with a problem definition
- Devise a minimal description (address) for any problem instance and sub-problem
- Define recurrence to specify the relationship of problems to sub-problems
  - i.e., Define the conceptual DAG on sub-problems
- Embed the DAG in a table
  - Use the index variables
    - 2-D case: index rows and columns
- Two possible strategies
  1. Fill in the sub-problem cells, proceeding from the smallest to the largest
  2. Draw the DAG in the table from the top problem down to the smallest sub-problems; solve the relevant sub-problems in their table cells, from smallest to largest. i.e., solve top problem recursively, using the table as a memory function

“Dynamic Programming”

- Very little to do with writing code
- Developed in the area of Operations Research
  - Conceived to optimally plan multi-stage processes
  - Then, “programming” = “planning”
- Solves an optimization problem
  - Described by recurrence relations among problems and sub-problems
  - Possibly time-variant
    - i.e. a dynamic system
  - In the CS setting, often time-invariant
- Coined by Bellman in the 1950s

End