CSE 417 Algorithms
Lecture 17, Winter 2020
Divide and Conquer
Dynamic Programming

Announcements

Divide and Conquer Algorithms

- Mergesort, Quicksort
- Strassen’s Algorithm
- Median
- Inversion counting
- Closest Pair Algorithm (2d)
- Integer Multiplication (Karatsuba’s Algorithm)

Integer Arithmetic

9715480283945084383094856701043643845790217965702956767
+ 124341983049057329675971989843092877957927759797

Runtime for standard algorithm to add two n digit numbers:

20950670930346809943185686868779409766717133476767930
+ 59201750917763470967767934292909701230956679993010921

Runtime for standard algorithm to multiply two n digit numbers:

Recursive Multiplication Algorithm (First attempt)

\[
x = x_1 2^{n/2} + x_0 \\
y = y_1 2^{n/2} + y_0 \\
xy = (x_1 2^{n/2} + x_0) (y_1 2^{n/2} + y_0) \\
\quad = x_1y_1 2^n + (x_1y_0 + x_0y_1)2^{n/2} + x_0y_0
\]

Recurrence:
Run time:

Simple algebra

\[
x = x_1 2^{n/2} + x_0 \\
y = y_1 2^{n/2} + y_0 \\
xy = x_1y_1 2^n + (x_1y_0 + x_0y_1) 2^{n/2} + x_0y_0
\]

\[
p = (x_1 + x_0)(y_1 + y_0) = x_1y_1 + x_1y_0 + x_0y_1 + x_0y_0
\]
Karatsuba’s Algorithm

Multiply \( n \)-digit integers \( x \) and \( y \)

Let \( x = x_1 \times 10^{n/2} + x_0 \) and \( y = y_1 \times 10^{n/2} + y_0 \)

Recursively compute

- \( a = x_1y_1 \)
- \( b = x_0y_0 \)
- \( p = (x_1 + x_0)(y_1 + y_0) \)

Return \( a2^n + (p - a - b)2^{n/2} + b \)

Recurrence: \( T(n) = 3T(n/2) + cn \)

Fast Integer Multiplication

- Grade School \( O(n^2) \)
- Karatsuba \( O(n^{1.58}) \)
- Toom-Cook \( O(n^{1.46}) \) [For 3 pieces]
  - \( O(n^{1+\epsilon}) \) [For \( k \) pieces]
- Schonhage-Strassen
  - Fast Fourier Transform based algorithm
  - \( O(n \log n \log \log n) \)
  - Becomes practical for \( \sim 25,000 \) digits

Dynamic Programming

- Weighted Interval Scheduling
- Given a collection of intervals \( I_1, \ldots, I_n \) with weights \( w_1, \ldots, w_n \), choose a maximum weight set of non-overlapping intervals

Optimality Condition

- \( \text{Opt}[j] \) is the maximum weight independent set of intervals \( I_1, I_2, \ldots, I_j \)
- \( \text{Opt}[j] = \max( \text{Opt}[j-1], w_j + \text{Opt}[p[j]]) \)
  - Where \( p[j] \) is the index of the last interval which finishes before \( I_j \) starts

Algorithm

\[
\text{MaxValue}(j) =
\begin{align*}
\text{if } j = 0 & \text{ return } 0 \\
\text{else} & \\
& \text{return } \max( \text{MaxValue}(j-1), w_j + \text{MaxValue}(p[j]))
\end{align*}
\]

Worst case run time: \( 2^n \)
A better algorithm

$M[j]$ initialized to -1 before the first recursive call for all $j$

$MaxValue(j) =$

- if $j = 0$ return 0;
- else if $M[j]! = -1$ return $M[j]$;
- else
  $M[j] = \max(\text{MaxValue}(j-1), w_j + \text{MaxValue}(p[j]));$
- return $M[j]$;

Iterative Algorithm

Express the $MaxValue$ algorithm as an iterative algorithm

$MaxValue$

Computing the solution

$Opt[j] = \max(Opt[j-1], w_j + Opt[p[j]])$

Dynamic Programming

- The most important algorithmic technique covered in CSE 421
- Key ideas
  - Express solution in terms of a polynomial number of sub problems
  - Order sub problems to avoid recomputation