Day 16 Notes

Zach Neveu

June 3, 2019

1 Agenda

- Knapsack branch and bound
- Project #4 Introduction
- Intro to Dynamic Programming
- Quiz Tomorrow
- HW due Thursday 6/6
- Project 4 due Monday 6/10
- Quiz Feedback

2 Knapsack Branch and Bound

- Recall example from last class
- Key Ideas
  - Branch on each item in knapsack
  - Estimate bound by filling remaining space with ratio of next highest item
  - This bound is somewhat loose - obviously not all items can be this good
- Improved bound: Add available items, when item won't fit, include the largest fraction of it that will!
- This is just like the LP bound on ILP!
- This knapsack problem, where items can be partially included is called **Fractional Knapsack**
- Solving fractional knapsack gives tight bound on solving 0-1 knapsack.

3 Project 4 Introduction

- 2 Part project
- Given header file for knapsack object
- implement `Knapsack::bound()` which finds the improved bound above

- Beware special cases!

- Implement `Knapsack::branchandbound()` searches for optimal solution using `bound()`
  - Decide how to branch on variables
  - Decide order to divide subproblems
  - Use knapsack object to store subproblems - each subproblem gets a knapsack
  - Use variable `num` to indicate how many variables have been fixed
  - Suggestion: use list structure (stl::deque) of subproblems. Choose item from deque, expand it, add new subproblems back to deque.
  - When feasible solns found keep track of the best one
  - 10 Minute limit

4 Dynamic Programming

- **Mysterious**
- NOT related to dynamic memory stuff
- Advanced algorithm design: make sure you are doing everything exactly once, not less, not more.
- **Dynamic Programming** is about not duplicating work
- Requires very specific properties. Rarely useful, but very good when it works.
- Revolves around Multiplying N matrices {a1, a2, a3, ..., aN}
- What order do multiplications go? For N=3, can use (A1*A2)A3 or A1(A2*A3).
- Order doesn't effect result, but does effect work required!
- Assume multiplying pxq and qxr matrices to get pxr.
- Final result is computing $p * r$ values
- Multiplications generally much slower than additions
- q products required to compute each entry of result
- pqr total products required - n^3 time, kinda slow

$\mathrm{A}_1 \rightarrow 10x100$
$A_2 \rightarrow 100x5$
$A_3 \rightarrow 5x50$
$(A_1 A_2) A_3 \rightarrow (10 * 100 * 5) + (10 * 5 * 50) = 7500$
$A_1 (A_2 A_3) \rightarrow (100 * 5 * 50) + (10 * 100 * 50) = 75,000$

- One answer is 100x harder to get!!

- How to minimize the work done?
- Number of ways to compute - huge for large N
- **Parenthesization** - number of ways to add parentheses to group values

$\mathrm{A}_1, A_2, A_3, A_4, \ldots, A_{n-1}, A_n$
$p_0 x p_1, p_1 x p_2, p_2 x p_3, \ldots, p_{n-1} x p_n$
$A_{i..j} = A_i A_{i+1} A_j \rightarrow PartialProduct$

- Assume an optimal parenthesization of some product has been found
  1. There must be a final product to compute
  2. Total cost of that optimal parenthesization = (cost to compute final product + price to compute LHS + price to compute RHS)
  3. **Optimal Substructure Property**: Optimal parenthesization of entire problem must contain optimal parenthesizations of subproblems. If LHS+RHS+FP is optimal, then LHS & RHS must both be optimal as well.
- Notation
  - Let $m[i,j]$ = Minimum # of multiplications needed to compute subproduct $A_{i..j}$
  - Looking for $m[1,n]$ eventually
  - Final multiplication: $A_{i..k} * A_{k+1..j}$.
  - $m[i,j] = m[i,k] + m[k+1,j] + p_{i-1} p_k p_j$
  - Problem: optimal k unknown...
  - Try all values of k and pick the best! Only $O(n)$
  - $m[i,j] = 0 if (i == j)$, else $argmin_k(m[i,k] + m[k+1,j] + p_{i-1} p_k p_j)$