6.046 Recitation 8: Bill Thies, Fall 2004

Dynamic Programming, Greedy Algorithms

Acknowledgements:

Parts of these notes were adapted from George Savvides and David Liben-Nowell.

My contact information:

Bill Thies

thies@mit.edu

Office hours: Sat 1-3pm, 36-153

Recitation website: http://cag.lcs.mit.edu/~thies/6.046/

Announcements:

- FINAL EXAM Mon 12/13, 9am-12pm, duPont
  - Please let me know ASAP if you have a conflict
- PS 5 due Mon
- PS 6 out Mon, due Wed 11/10

Today:

- Examples of Dynamic Programming and Greedy algorithms

Chained Matrix Multiplication

A: m x n
B: n x p
AB: m x p

Time to compute AB? (n)(m)(p)
<< have m x p entries of AB, each needs n-element dot product >>
- this is using naïve algorithm; could do slightly better with Strassen, etc.

Chained Matrix Multiplication: << multiply a sequence of matrices >>
<< the evaluation order makes a difference >>

for example, consider A B C

Dimensions:
A: 1 x 10
B: 10 x 5
C: 5 x 10

AB: 1 x 5
BC: 10 x 10

(AB)C costs (1)(10)(5) + (1)(5)(10) = 100
A(BC) costs (10)(5)(10) + (1)(10)(10) = 600

General Problem

Given: Matrices $M_1, M_2, \ldots, M_n$
Matrix M_i has dimensions $d_i \times d_{i+1}$

Goal: parenthesize to minimize <u>total</u> cost of multiplication

How many different parenthesizations P(n)?

$P(n) =$ 1 if n=1
$\sum_{i=1}^{n-1} P(k)\, P(n-k)$ otherwise

Can show $P(n) > 2^n$ using substitution method
RHS is $\sum_{i=1}^{n-1} 2^k\, 2^{n-k} = \sum_{i=1}^{n-1} 2^n > 2^n$

By the way:
P(n) is closely related to the n'th Catalan number, = (2n choose n) / (n+1)
Catalan (n) = number of binary trees with n nodes

Dynamic Programming

1. Optimal substructure
2. Overlapping sub-problems

How to show optimal sub-structure in chained matrix multiplication?

Want to show, for example:

if $((M_1\ M_2)\ M_3)\ (M_4\ M_5)$ is optimal order,

then both $(M_1\ M_2)\ M_3$ and $M_4\ M_5$ are optimal orders of sub-problems

Proof: "cut-and-paste"

Assume that $M_1\ (M_2\ M_3)$ was better order for first sub-problem. Then we could cut-and-paste that solution into the overall solution to improve the global optimum. But this contradicts the fact that $((M_1\ M_2)\ M_3)\ (M_4\ M_5)$ is optimal order, because the overall cost is the SUM of the cost of the left, the cost of the right, and the cost of the multiply. Changing the order on the left does not effect the cost of the multiply (it does not change resulting dimensions) or the cost of the right, so its only effect on the overall cost is to decrease the cost of the left. Thereby decreasing the cost of the overall solution, which is a contradiction.

Formulating Optimal Solution

Let opt[i,j] = min cost for $M_i \ldots M_j$

# of sub-problems? (n choose 2) + n = $\Theta(n^2)$

How to calculate cost? Consider that we split before position k:

$(M_i \ldots M_{k-1})(M_k \ldots M_j)$

dims: $d_i \times d_k \quad | \quad d_k \times d_{j+1}$

So Cost(i, j, k) = opt[i, k-1] + opt[k, j] + $d_i\, d_k\, d_{j+1}$

But we need to choose $k \in [i, j]$…

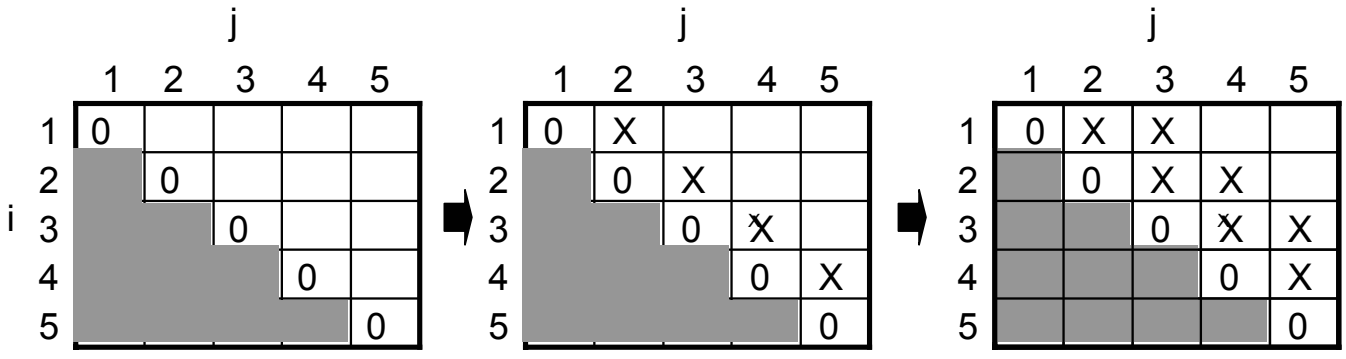
$$\text{opt}[i,j] = \begin{cases} 0 \text{ if } i=j \\ \min_{i < k \le j} \text{opt}[i, k-1] + \text{opt}[k, j] + d_i\, d_k\, d_{j+1} \text{ otherwise} \end{cases}$$

Final answer? Stored in opt[1,n]

Evaluation order? Opt[1, 1], Opt[2, 2], … [evaluate for interval (i, i)]
Opt[1, 2], Opt[2, 3] … [evaluate for interval (i, i+1)]

Evaluating in this "bottom-up" order, we guarantee that we calculate each entry opt[i,k-1] and opt[k,j] that opt[i,j] depends on, before entry opt[i,j] itself.

Graphical version of OPT being filled in:



Formulating Optimal Solution (cont'd)

Runtime? = # sub-problems x time / subproblem

= $\Theta(n^2)$ * $O(n)$ [abuse of notation]

= $O(n^3)$

How do you construct set of parentheses from optimal cost?

Second phase does a "traceback" through the optimal costs, reconstructing the k that was chosen at each step.

Can optimize this process by storing k as you compute each opt[i,j]. Let s[i,j] = k that was chosen as minimum for opt[i,j].

Then, for example, if s[1, 100] = 17, the split is as follows:

$(M_1 \ldots M_{16})\ (M_{17} \ldots M_{100})$

Can use s[1,16] and s[17,100] to determine next ones.

Example Greedy Algorithm

Professor Midas drives an automobile cross-country.

- His gas tank can hold enough gas to drive k miles.
- Gas stations $g_1, g_2, \ldots g_n$ are separated by distances $d_1, d_2, \ldots, d_n$
- Whenever he stops, he fills up his tank

Which stations should he stop at so as to minimize the total number of stops?

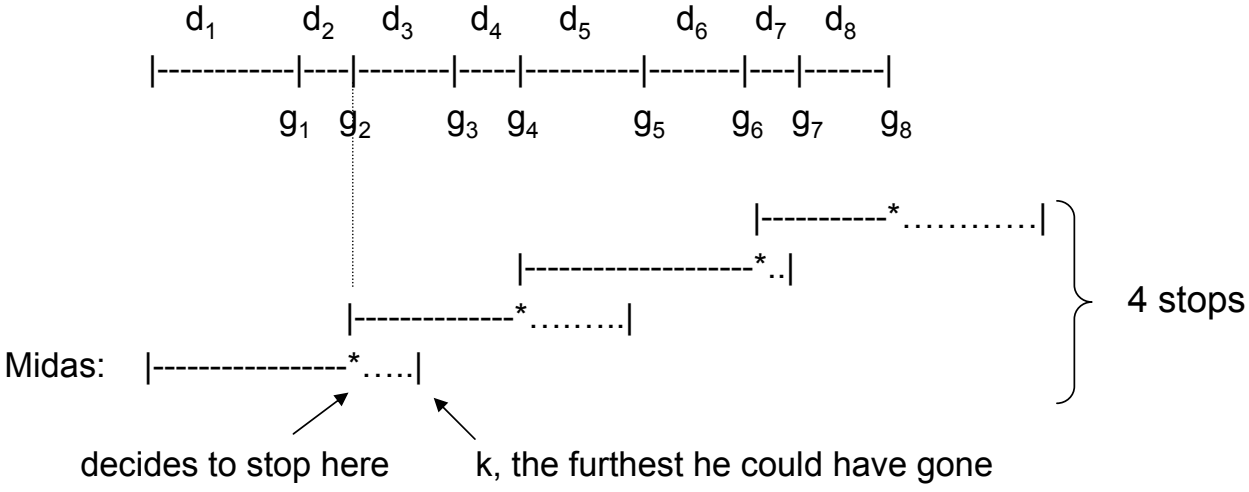
Use greedy algorithm. Properties of greedy algorithm:

1. Optimal substructure
2. Greedy choice property: locally optimal choice is also globally optimal

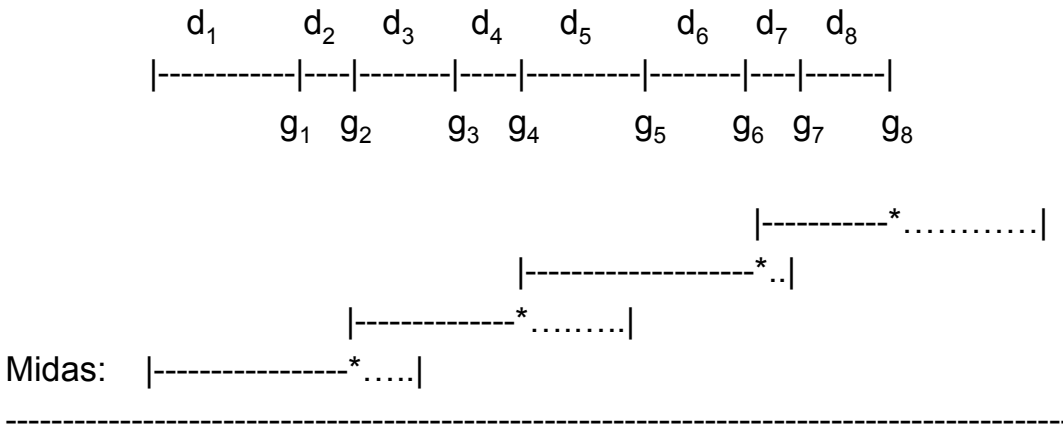
Greedy alg. for Midas: Drive car as far as possible. Only stops for gas if he could not make it to the next stop otherwise.

Clearly, $k > \max_i d_i$, as otherwise it would be impossible to finish

Gas Stations:



Optimality of Greedy Algorithm



Proof of why greedy is optimal: induction on D(i), the maximum distance that can be covered in i stops. Let D_{OPT} denote the optimal and D_{Midas} denote the greedy.

Base case: $D_{Midas}(1) \geq D_{OPT}(1)$ because Midas travels as far as possible on the first gas tank.

Inductive step: Assume $D_{Midas}(m) \geq D_{OPT}(m)$. Then on step m+1, both OPT and Midas can cover up to k additional miles. Midas stops at the last gas station before $D_{Midas}(m) + k$. On this step, OPT started no further than Midas and travelled the same distance, so OPT can not reach any further gas station. Therefore $D_{Midas}(m+1) \geq D_{OPT}(m+1)$.

Since Midas can always cover at least as much distance as OPT in m steps, then Midas is guaranteed to use no more stops than OPT on any fixed distance.

<< in class, formulated with cut-and-paste, but this is more natural >>

<u>Extensions:</u>

- In 1pm section, considered price / gallon at each node and minimizing cost instead of # stops. Can also solve greedily by buying enough gas to travel to next CHEAPEST station. If no such station in range, fill up tank.
- In 1pm section, also considered jointly minimizing # stops and price. I don't think there is greedy algorithm for this, or even polynomial-time algorithm. Left it open!

Other good examples of greedy / dynamic programming (sketch)

GREEDY:

- Huffman coding for data compression (CLRS 16.3)
  - (This is cool, I had planned to do this in class, but didn't have time)
- Kruskal's algorithm for Minimum Spanning Trees (CLRS p. 568)
- Minimal number of coins to give change, using U.S. denominations
- There are n points on real line, how many unit intervals do you need to cover them?

DYNAMIC PROGRAMMING:

- Optimal arrangement of keys in binary search tree (CLRS 15.5)
- Minimal number of coins to give change, using arbitrary denominations
  - E.g, denonimations 1, 10, 25. Then 30 = 10 + 10 + 10 uses only 3 coins, but 30 = 25 + 1 + 1 + 1 + 1 + 1 uses 6 coins. So need DP, not just greedy.

Also several examples of both greedy and dynamic programming in computing shortest paths information… will cover next week in class.