Carlos Mc Gregor c.mcgregormuro@mail.utoronto.ca

CSC373S: Algorithm Design, Analysis & Complexity

Lecture 10

Friday January 27, 2017
based on notes by Denis Pankratov

Dynamic Programming (*continue*)

Chain Matrix Multiplication

Define: cost of multiplyling two matrices A_1 of dimension $n \times p$ and A_2 of dimension $p \times m$ as npm.

Remember: Matrix multiplication is associative!

$$(A \times B) \times C = A \times (B \times C)$$

Overall cost of matrix multiplication (of n matrices) depends on parenthesization.

Example: Given the following matrices w/ their respective... dimensions

Matrix name	Dimension
A	50×20
B	20×1
C	1×10
D	10×100

We have to come up with a way to optimize the multiplication of these 4 matrices. The key is working with the values $50, 20, 1, 10, 100$ in the most efficient way.

Keep in mind that:

Parenthesis	Overall Cost
$A \times ((B \times C) \times D)$	$20 \times 1 \times 10 + 20 \times 10 \times 100 + 50 \times 20 \times 100 = 120,200$
$(A \times (B \times C)) \times D$	$20 \times 1 \times 10 + 50 \times 20 \times 10 + 50 \times 10 \times 100 = 60,200$
$(A \times B) \times (C \times D)$	$50 \times 20 \times 1 + 1 \times 10 \times 100 + 50 \times 1 \times 100 = 7,000$

Input: D - array of $n+1$ positive integers (0-based indexing), which represents:

$$D[0] \times D[1] - \text{dimension of } A_1$$

$$D[1] \times D[2] - \text{dimension of } A_2$$

$$\vdots$$

$$D[n-1] \times D[n] - \text{dimension of } A_n$$

Output: Optimal (*as in, with minimum overall cost* parenthesization.

Rough Approach: Asume that we already have an optimal solution of the form

$$(A_1 \times A_2 \times \cdots \times A_k) \times (A_{k+1} \times \cdots \times A_n)$$

So that the first interval has a dimension of $D[0] \times D[k]$, while the second interval has a dimension of $D[k] \times D[n]$. k is used to split the left from the right part of our multiplication.

Semantic array:
$C[i,j]$ = minimum overall cost to multiply matrices $A_i \times \cdots \times A_j$, $D[i-1], D[i], \ldots, D[j]$.

Computational array:
$i < j, C[i,j] = \min_{i \leq k < j}\{C[i,k] + C[k+1,j] + D[i-1] \cdot D[k] \cdot D[j]\}$.

Base case: $C[i,i] = 0$

Correctness (equivalence):

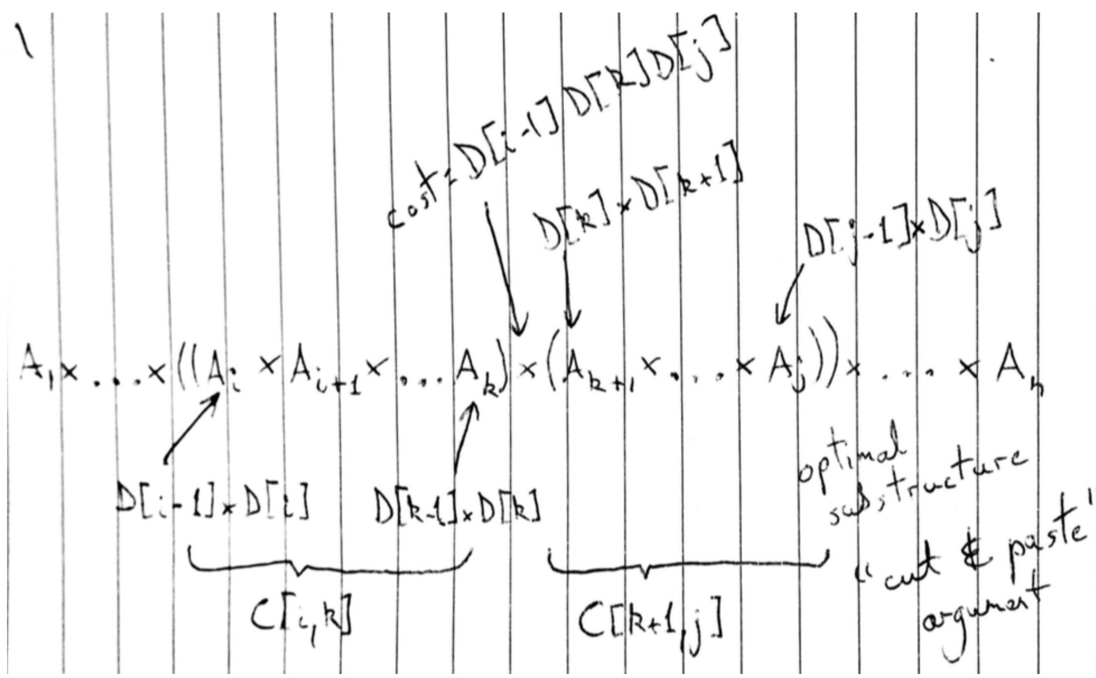

Figure 1: Rough visual representation of correctness

Algorithm: Pseudocode to compute the *value* of optimal

```
def CMM(D, n):
    initialize C array of size n * n (1-based indexing)
```

Carlos Mc Gregor c.mcgregormuro@mail.utoronto.ca

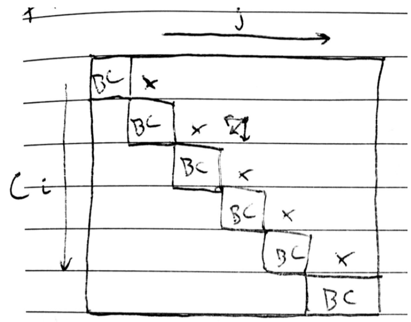

Figure 2: Visual representation of optimal path: fill in the entries by increasing $j - i$

```
for i = 1 to n:
  C[i,i] = 0
for l = 1 to n - 1:
  for i = 1 to n - l:
    j = i + l
    C[i,j] = float('inf')
    for k = i to j - 1:
      C[i, j] = min(C[i, j],
                     C[i, k] + C[k + 1, j] + D[i = 1] * D[k] * D[j])
return C[1, n]
```

<u>Exercise:</u> Add book-keeping to compute an optimal parethesization

Max Independent Set in Trees

<u>Definition:</u> Let $T = (V, E)$ be a tree $S \subseteq V$ is an *independent* set if there are no edges in T between vertices in S. (*see Figure 3 below*).

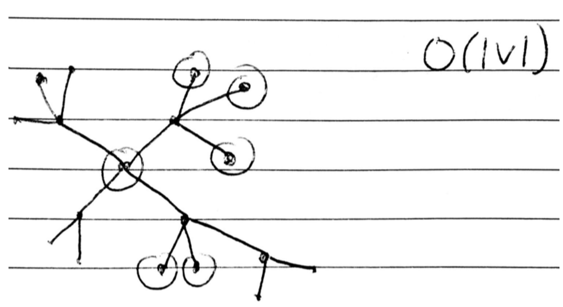

Figure 3: Example of $T = (V, E)$ with target runtime of $\mathcal{O}(|V|)$

Input: $T = (V, E)$ - tree (rooted), r - root

Output: S - max. independent set

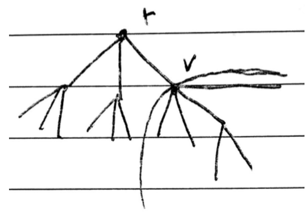

Figure 4: Visualization of r, v and max. independent set

Semantic array:
$\forall v \in V, C[v] =$ size of max. independent set of the subtree hanging from the vertex v

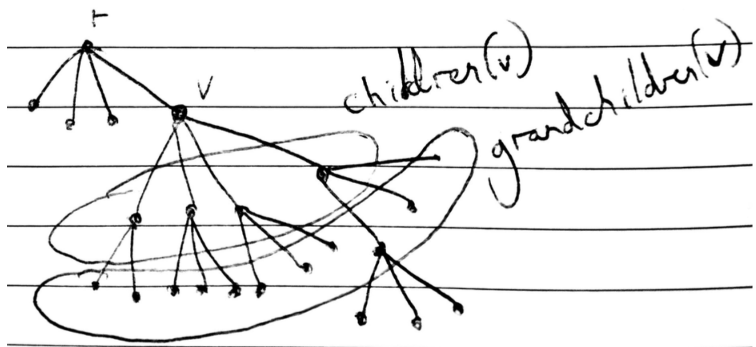

Figure 5: Visualization of v with Children(v) and GrandChildren(v)

Computational array:
$C[v] = \max\{1 + \sum_{u \in \text{GrandChildren}(v)} C[u], \sum_{u \in \text{Children}(v)} C[u])$, where the first element in min is $v \in OPT$, and the second is $v \notin OPT$.

<u>Exercise:</u>

1. finish proof of correctness
2. *pseudocode:* recursion and memoization

Runtime should be $\mathcal{O}(|V|)$