Matrix Chain Multiplication

- Let A be a $p \times q$ matrix
- Let B be a $q \times r$ matrix
- $AB = C$ C is $p \times r$
- To calculate C we do pqr scalar multiplications

Given a sequence of n matrices $A_1 A_2 A_3 \ldots A_n$ to be multiplied together, where A_i has dimensions $P_{i-1} \times P_i$, determine the minimum number of scalar multiplications required.

- 3 matrices A, B, C with dimensions $p \times q$, $q \times r$, $r \times s$
- Since Matrix multiplication is Associative we get the same result whether we compute $(AB)C$ or $A(BC)$
- The number of scalar multiplications can vary

$$(AB)C \longrightarrow pqr + prs$$

$$A(BC) \longrightarrow qrs + pqs$$

$p = 10$
$q = 100$
$r = 5$
$s = 50$

$(AB)C \rightarrow 7,500$

$A(BC) \rightarrow 75,000$

Full Parenthesization

- $n = 1$ or $n = 2$ there is only 1
- $n = 3$ there are 2
- $n = 4$

$(((A_1 A_2) A_3) A_4)$

$((A_1 A_2)(A_3 A_4))$

$((A_1 (A_2 A_3)) A_4)$

$(A_1 ((A_2 A_3) A_4))$

$(A_1 (A_2 (A_3 A_4)))$

$$P(1) = 1$$

$$P(n) = \sum_{k=1}^{n-1} P(k) P(n-k)$$

```
RECURSIVEMATRIXCHAIN(p, i, j)
1  if i == j
2        return 0
3  m = ∞
4  for k = i to j − 1
5        q = RECURSIVEMATRIXCHAIN(p, i, k)
         +RECURSIVEMATRIXCHAIN(p, k + 1, j) + p_{i−1} p_k p_j
6        m = min(m, q)
7  return m[i, j]
```

Complexity

$$T(1) = 1$$

$$T(n) = \sum_{k=1}^{n-1} T(k)\,T(n-k)$$

$$T(n) \in \Omega(2^n)$$

Proof

- Assume that $T(k) \geq c\,2^k \quad \forall\, k < n$

- for $n > 1$

$$T(n) = \sum_{k=1}^{n-1} T(k)\,T(n-k)$$

$$\geq \sum_{k=1}^{n-1} c\,2^k\, c\,2^{n-k}$$

$$= \sum_{k=1}^{n-1} \frac{c\,2^k\, c\,2^n}{2^k}$$

$$= \sum_{k=1}^{n-1} c^2\,2^n = (n-1)\,c^2\,2^n \geq c\,2^n$$

$$2^{-k} = \frac{1}{2^k}$$

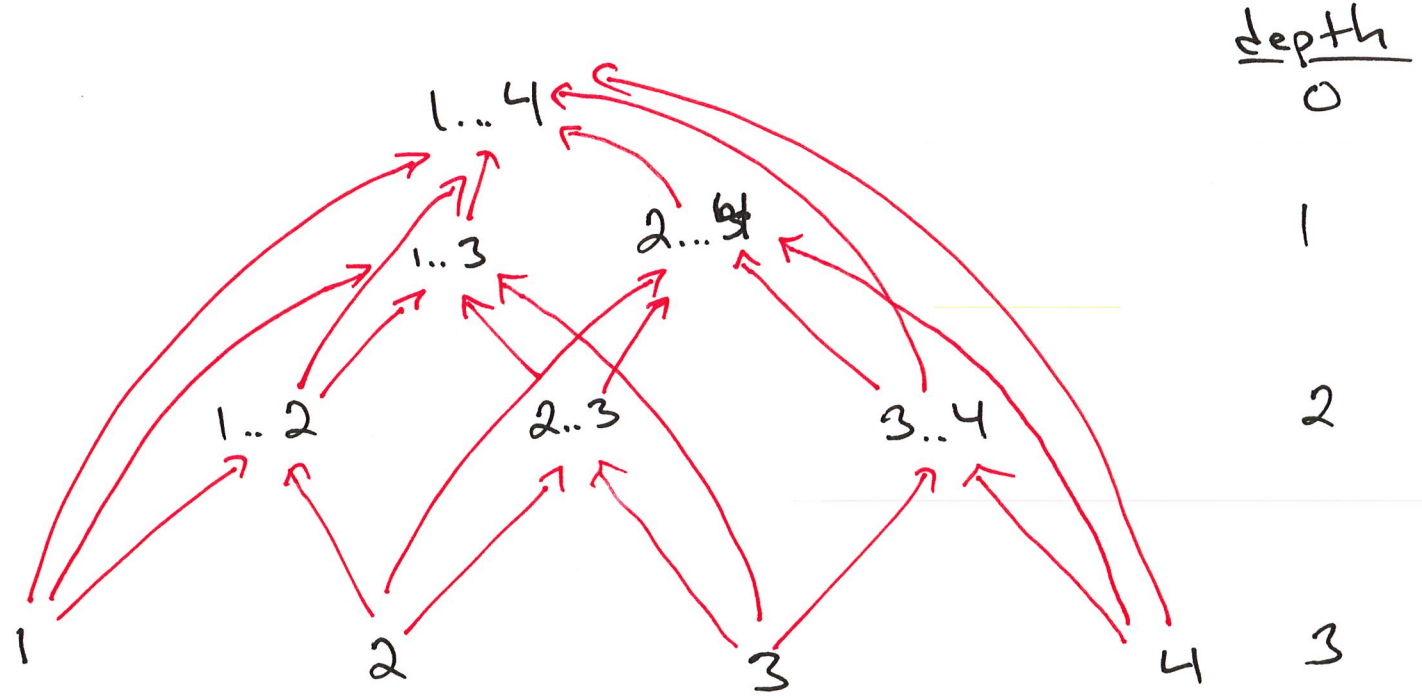
$$2^{n-k} = \frac{2^n}{2^k}$$

Optimal Substructure

- Let $A_{i..j}$ where $i \leq j$ be the matrix that results from multiplying $A_i A_{i+1} \ldots A_j$
- if $i < j$ we must split between A_k and A_{k+1} for some k $i < k < j$
  - that is for some k we must first compute $A_{i..k}$ and $A_{k+1..j}$ and then multiply them together to compute $A_{i..j}$
  - the cost of parenthesizing in this way is the cost to compute $A_{i..k}$ ~~plus~~ plus the cost to compute $A_{k+1..j}$ plus the cost to multiply them.

- suppose that to optimally parenthesize $A_i A_{i+1} \dots A_j$ we must split between A_k and A_{k+1}
- the way we parenthesize $A_i A_{i+1} \dots A_k$ within this must also be optimal
- If there was a less costly way to parenthesize $A_i \dots A_k$ we could substitute that into our parenthesization of $A_i A_{i+1} \dots A_j$ to produce a less costly one.

DAG $A_1\ A_2\ A_3\ A_4$



- The DAG has height $n-1$
- at each depth d $0 \leq d < n$ there are $d+1$ nodes which must each do $n-d-1$ units of work
- overall work: $\sum_{d=0}^{n-1} (d+1)(n-d-1) \in \Theta(n^3)$

```
MEMOIZEDMATRIXCHAIN(p, i, j)
1  if i == j
2        return 0
3  if m[i, j] == NIL
4        m[i, j] = ∞
5        for k = i to j − 1
6              q = MEMOIZEDMATRIXCHAIN(p, i, k)
                   +MEMOIZEDMATRIXCHAIN(p, k + 1, j) + p_{i−1} p_k p_j
7              m[i, j] = min(m[i, j], q)
8  return m[i, j]
```

```
MATRIX-CHAIN-ORDER(p)
 1  n = p.length − 1
 2  let m[1..n, 1..n] be a new Table
 3  for i = 1 to n
 4      m[i, i] = 0
 5  for l = 2 to n // l is the chain length
 6      for i = 1 to n − l + 1
 7          j = i + l − 1
 8          m[i, j] = ∞
 9          for k = i to j − 1
10              q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
11              if q < m[i, j]
12                  m[i, j] = q
13  return m[1, n]
```

Minimum Edit Distance

- A measure of String similarity
- It is calculated by counting the minimum number of edits/operations required to transform one string into another.

Operations

- insertion
- deletion
- substitution

S = ABCADA

T = ABADC

ABCADA → ABADC : 5 operations

ABCADA → ABADC : 2 operations

Spelling Correction

Some user typed Graffe.

- graf
- graft
- grail
- ~~giraffe~~
- giraffe

S= INTENTION

T= EXECUTION

MED(S,T) = 5

~~I N T E N T I O N~~

I	N	T	E	—	N	T	I	O	N
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
—	E	X	E	C	U	T	I	O	N

```
MED(S, T)
 1  if |S| == 0
 2        return |T|
 3  elseif |T| == 0
 4        return |S|
 5  else
 6        write S = sS' and T = tT'
 7        if s == t
 8              return MED(S', T')
 9        else
10              d = MED(S', T)
11              e = MED(S, T')
12              f = MED(S', T')
13              return 1 + min(d, e, f)
```

Worst Case Complexity

- Let $m = |S|$ and $n = |T|$

$$T(m,n) = T(m-1,n) + T(m,n-1) + T(m-1,n-1) + 1$$

$$T(0,n) = 1$$

$$T(m,0) = 1$$

- Each $T(m,n)$ splits into 3 paths,

Since each recursive call runs in $O(1)$-time we just need to count the number of nodes.

- The shortest path has length $\min(m,n)$

  - so the tree has at least

    $3^{\min(m,n)}$ nodes

    $\Omega(3^{\min(m,n)})$

- The longest path has length $m+n-1$

  - so the tree has at most

    3^{m+n-1} nodes

    $O(3^{m+n-1})$