Matrix Multiplication

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Programming, Data Structures and Algorithms using Python

Week 9

Multiplying matrices

- Multiply matrices A, B
  - $AB[i,j] = \sum_{k=0}^{n-1} A[i,k]B[k,j]$

Multiplying matrices

- Multiply matrices A, B
  - $AB[i,j] = \sum_{k=0}^{n-1} A[i,k]B[k,j]$
- Dimensions must be compatible
  - $A : m \times n$, $B : n \times p$
  - $AB : m \times p$

Multiplying matrices

- Multiply matrices A, B
  - $AB[i,j] = \sum_{k=0}^{n-1} A[i,k]B[k,j]$
- Dimensions must be compatible
  - $A : m \times n$, $B : n \times p$
  - $AB : m \times p$
- Computing each entry in AB is $O(n)$

Multiplying matrices

- Multiply matrices A, B
  - $AB[i,j] = \sum_{k=0}^{n-1} A[i,k]B[k,j]$
- Dimensions must be compatible
  - $A : m \times n$, $B : n \times p$
  - $AB : m \times p$
- Computing each entry in AB is $O(n)$
- Overall, computing AB is $O(mnp)$

Multiplying matrices

- Multiply matrices A, B
  - $AB[i,j] = \sum_{k=0}^{n-1} A[i,k]B[k,j]$
- Dimensions must be compatible
  - $A : m \times n$, $B : n \times p$
  - $AB : m \times p$
- Computing each entry in AB is $O(n)$
- Overall, computing AB is $O(mnp)$
- Matrix multiplication is associative
  - $ABC = (AB)C = A(BC)$

Multiplying matrices

- Multiply matrices A, B
  - $AB[i,j] = \sum_{k=0}^{n-1} A[i,k]B[k,j]$
- Dimensions must be compatible
  - $A : m \times n$, $B : n \times p$
  - $AB : m \times p$
- Computing each entry in AB is $O(n)$
- Overall, computing AB is $O(mnp)$
- Matrix multiplication is associative
  - $ABC = (AB)C = A(BC)$
  - Bracketing does not change answer

Multiplying matrices

- Multiply matrices A, B
  - $AB[i,j] = \sum_{k=0}^{n-1} A[i,k]B[k,j]$
- Dimensions must be compatible
  - $A : m \times n$, $B : n \times p$
  - $AB : m \times p$
- Computing each entry in AB is $O(n)$
- Overall, computing AB is $O(mnp)$
- Matrix multiplication is associative
  - $ABC = (AB)C = A(BC)$
  - Bracketing does not change answer
  - ... but can affect the complexity!

Multiplying matrices

- Multiply matrices A, B
  - $AB[i,j] = \sum_{k=0}^{n-1} A[i,k]B[k,j]$
- Dimensions must be compatible
  - $A : m \times n$, $B : n \times p$
  - $AB : m \times p$
- Computing each entry in AB is $O(n)$
- Overall, computing AB is $O(mnp)$
- Matrix multiplication is associative
  - $ABC = (AB)C = A(BC)$
  - Bracketing does not change answer
  - ... but can affect the complexity!

- Let $A : 1 \times 100$, $B : 100 \times 1$, $C : 1 \times 100$

Multiplying matrices

- Multiply matrices A, B
  - $AB[i,j] = \sum_{k=0}^{n-1} A[i,k]B[k,j]$
- Dimensions must be compatible
  - $A : m \times n$, $B : n \times p$
  - $AB : m \times p$
- Computing each entry in AB is $O(n)$
- Overall, computing AB is $O(mnp)$
- Matrix multiplication is associative
  - $ABC = (AB)C = A(BC)$
  - Bracketing does not change answer
  - ...but can affect the complexity!

- Let $A : 1 \times 100$, $B : 100 \times 1$, $C : 1 \times 100$
- Computing $A(BC)$

Multiplying matrices

- Multiply matrices A, B
  - $AB[i,j] = \sum_{k=0}^{n-1} A[i,k]B[k,j]$
- Dimensions must be compatible
  - $A : m \times n$, $B : n \times p$
  - $AB : m \times p$
- Computing each entry in AB is $O(n)$
- Overall, computing AB is $O(mnp)$
- Matrix multiplication is associative
  - $ABC = (AB)C = A(BC)$
  - Bracketing does not change answer
  - ...but can affect the complexity!
- Let $A : 1 \times 100$, $B : 100 \times 1$, $C : 1 \times 100$
- Computing $A(BC)$
  - $BC : 100 \times 100$, takes $100 \cdot 1 \cdot 100 = 10000$ steps to compute

Multiplying matrices

- Multiply matrices A, B
  - $AB[i,j] = \sum_{k=0}^{n-1} A[i,k]B[k,j]$
- Dimensions must be compatible
  - $A : m \times n$, $B : n \times p$
  - $AB : m \times p$
- Computing each entry in AB is $O(n)$
- Overall, computing AB is $O(mnp)$
- Matrix multiplication is associative
  - $ABC = (AB)C = A(BC)$
  - Bracketing does not change answer
  - . . . but can affect the complexity!
- Let $A : 1 \times 100$, $B : 100 \times 1$, $C : 1 \times 100$
- Computing $A(BC)$
  - $BC : 100 \times 100$, takes $100 \cdot 1 \cdot 100 = 10000$ steps to compute
  - $A(BC) : 1 \times 100$, takes $1 \cdot 100 \cdot 100 = 10000$ steps to compute

Multiplying matrices

- Multiply matrices A, B
  - $AB[i,j] = \sum_{k=0}^{n-1} A[i,k]B[k,j]$
- Dimensions must be compatible
  - $A : m \times n$, $B : n \times p$
  - $AB : m \times p$
- Computing each entry in AB is $O(n)$
- Overall, computing AB is $O(mnp)$
- Matrix multiplication is associative
  - $ABC = (AB)C = A(BC)$
  - Bracketing does not change answer
  - . . . but can affect the complexity!

- Let $A : 1 \times 100$, $B : 100 \times 1$, $C : 1 \times 100$
- Computing $A(BC)$
  - $BC : 100 \times 100$, takes $100 \cdot 1 \cdot 100 = 10000$ steps to compute
  - $A(BC) : 1 \times 100$, takes $1 \cdot 100 \cdot 100 = 10000$ steps to compute
- Computing $(AB)C$

Multiplying matrices

- Multiply matrices A, B
  - $AB[i,j] = \sum_{k=0}^{n-1} A[i,k]B[k,j]$
- Dimensions must be compatible
  - $A : m \times n$, $B : n \times p$
  - $AB : m \times p$
- Computing each entry in AB is $O(n)$
- Overall, computing AB is $O(mnp)$
- Matrix multiplication is associative
  - $ABC = (AB)C = A(BC)$
  - Bracketing does not change answer
  - ...but can affect the complexity!
- Let $A : 1 \times 100$, $B : 100 \times 1$, $C : 1 \times 100$
- Computing $A(BC)$
  - $BC : 100 \times 100$, takes $100 \cdot 1 \cdot 100 = 10000$ steps to compute
  - $A(BC) : 1 \times 100$, takes $1 \cdot 100 \cdot 100 = 10000$ steps to compute
- Computing $(AB)C$
  - $AB : 1 \times 1$, takes $1 \cdot 100 \cdot 1 = 100$ steps to compute

Multiplying matrices

- Multiply matrices A, B
  - $AB[i,j] = \sum_{k=0}^{n-1} A[i,k]B[k,j]$
- Dimensions must be compatible
  - $A : m \times n$, $B : n \times p$
  - $AB : m \times p$
- Computing each entry in AB is $O(n)$
- Overall, computing AB is $O(mnp)$
- Matrix multiplication is associative
  - $ABC = (AB)C = A(BC)$
  - Bracketing does not change answer
  - ...but can affect the complexity!

- Let $A : 1 \times 100$, $B : 100 \times 1$, $C : 1 \times 100$
- Computing $A(BC)$
  - $BC : 100 \times 100$, takes $100 \cdot 1 \cdot 100 = 10000$ steps to compute
  - $A(BC) : 1 \times 100$, takes $1 \cdot 100 \cdot 100 = 10000$ steps to compute
- Computing $(AB)C$
  - $AB : 1 \times 1$, takes $1 \cdot 100 \cdot 1 = 100$ steps to compute
  - $(AB)C : 1 \times 100$, takes $1 \cdot 1 \cdot 100 = 100$ steps to compute

Multiplying matrices

- Multiply matrices A, B
  - $AB[i,j] = \sum_{k=0}^{n-1} A[i,k]B[k,j]$
- Dimensions must be compatible
  - $A : m \times n$, $B : n \times p$
  - $AB : m \times p$
- Computing each entry in AB is $O(n)$
- Overall, computing AB is $O(mnp)$
- Matrix multiplication is associative
  - $ABC = (AB)C = A(BC)$
  - Bracketing does not change answer
  - . . . but can affect the complexity!

- Let $A : 1 \times 100$, $B : 100 \times 1$, $C : 1 \times 100$
- Computing $A(BC)$
  - $BC : 100 \times 100$, takes $100 \cdot 1 \cdot 100 = 10000$ steps to compute
  - $A(BC) : 1 \times 100$, takes $1 \cdot 100 \cdot 100 = 10000$ steps to compute
- Computing $(AB)C$
  - $AB : 1 \times 1$, takes $1 \cdot 100 \cdot 1 = 100$ steps to compute
  - $(AB)C) : 1 \times 100$, takes $1 \cdot 1 \cdot 100 = 100$ steps to compute
- 20000 steps vs 200 steps!

Multiplying matrices

- Multiply matrices A, B
  - $AB[i,j] = \sum_{k=0}^{n-1} A[i,k]B[k,j]$
- Dimensions must be compatible
  - $A : m \times n$, $B : n \times p$
  - $AB : m \times p$
- Computing each entry in AB is $O(n)$
- Overall, computing AB is $O(mnp)$
- Matrix multiplication is associative
  - $ABC = (AB)C = A(BC)$
  - Bracketing does not change answer
  - . . . but can affect the complexity!

Multiplying matrices

- Multiply matrices A, B
  - $AB[i,j] = \sum_{k=0}^{n-1} A[i,k]B[k,j]$
- Dimensions must be compatible
  - $A : m \times n$, $B : n \times p$
  - $AB : m \times p$
- Computing each entry in AB is $O(n)$
- Overall, computing AB is $O(mnp)$
- Matrix multiplication is associative
  - $ABC = (AB)C = A(BC)$
  - Bracketing does not change answer
  - . . . but can affect the complexity!
- Given n matrices $M_0 : r_0 \times c_0$, $M_1 : r_1 \times c_1$, . . . , $M_{n-1} : r_{n-1} \times c_{n-1}$

Multiplying matrices

- Multiply matrices A, B
  - $AB[i,j] = \sum_{k=0}^{n-1} A[i,k]B[k,j]$
- Dimensions must be compatible
  - $A : m \times n$, $B : n \times p$
  - $AB : m \times p$
- Computing each entry in AB is $O(n)$
- Overall, computing AB is $O(mnp)$
- Matrix multiplication is associative
  - $ABC = (AB)C = A(BC)$
  - Bracketing does not change answer
  - ...but can affect the complexity!
- Given n matrices $M_0 : r_0 \times c_0$, $M_1 : r_1 \times c_1$, ..., $M_{n-1} : r_{n-1} \times c_{n-1}$
  - Dimensions match: $r_j = c_{j-1}$, $0 < j < n$

Multiplying matrices

- Multiply matrices A, B
  - $AB[i,j] = \sum_{k=0}^{n-1} A[i,k]B[k,j]$
- Dimensions must be compatible
  - $A : m \times n$, $B : n \times p$
  - $AB : m \times p$
- Computing each entry in AB is $O(n)$
- Overall, computing AB is $O(mnp)$
- Matrix multiplication is associative
  - $ABC = (AB)C = A(BC)$
  - Bracketing does not change answer
  - . . . but can affect the complexity!
- Given n matrices $M_0 : r_0 \times c_0$, $M_1 : r_1 \times c_1$, . . . , $M_{n-1} : r_{n-1} \times c_{n-1}$
  - Dimensions match: $r_j = c_{j-1}$, $0 < j < n$
  - Product $M_0 \cdot M_1 \cdots M_{n-1}$ can be computed

Multiplying matrices

- Multiply matrices A, B
  - $AB[i,j] = \sum_{k=0}^{n-1} A[i,k]B[k,j]$
- Dimensions must be compatible
  - $A : m \times n$, $B : n \times p$
  - $AB : m \times p$
- Computing each entry in AB is $O(n)$
- Overall, computing AB is $O(mnp)$
- Matrix multiplication is associative
  - $ABC = (AB)C = A(BC)$
  - Bracketing does not change answer
  - ...but can affect the complexity!

- Given n matrices $M_0 : r_0 \times c_0$, $M_1 : r_1 \times c_1$, ..., $M_{n-1} : r_{n-1} \times c_{n-1}$
  - Dimensions match: $r_j = c_{j-1}$, $0 < j < n$
  - Product $M_0 \cdot M_1 \cdots M_{n-1}$ can be computed
- Find an optimal order to compute the product
  - Multiply two matrices at a time
  - Bracket the expression optimally

Inductive structure

- Final step combines two subproducts
  $(M_0 \cdot M_1 \cdots M_{k-1}) \cdot (M_k \cdot M_{k+1} \cdots M_{n-1})$
  for some $0 < k < n$

Inductive structure

- Final step combines two subproducts
  $(M_0 \cdot M_1 \cdots M_{k-1}) \cdot (M_k \cdot M_{k+1} \cdots M_{n-1})$
  for some $0 < k < n$

- First factor is $r_0 \times c_{k-1}$, second is
  $r_k \times c_{n-1}$, where $r_k = c_{k-1}$

Inductive structure

- Final step combines two subproducts $(M_0 \cdot M_1 \cdots M_{k-1}) \cdot (M_k \cdot M_{k+1} \cdots M_{n-1})$ for some $0 < k < n$
- First factor is $r_0 \times c_{k-1}$, second is $r_k \times c_{n-1}$, where $r_k = c_{k-1}$
- Let $C(0, n{-}1)$ denote the overall cost

Inductive structure

- Final step combines two subproducts $(M_0 \cdot M_1 \cdots M_{k-1}) \cdot (M_k \cdot M_{k+1} \cdots M_{n-1})$ for some $0 < k < n$
- First factor is $r_0 \times c_{k-1}$, second is $r_k \times c_{n-1}$, where $r_k = c_{k-1}$
- Let $C(0, n-1)$ denote the overall cost
- Final multiplication is $O(r_0 r_k c_{n-1})$

Inductive structure

- Final step combines two subproducts
  $(M_0 \cdot M_1 \cdots M_{k-1}) \cdot (M_k \cdot M_{k+1} \cdots M_{n-1})$
  for some $0 < k < n$
- First factor is $r_0 \times c_{k-1}$, second is $r_k \times c_{n-1}$, where $r_k = c_{k-1}$
- Let $C(0, n-1)$ denote the overall cost
- Final multiplication is $O(r_0 r_k c_{n-1})$
- Inductively, costs of factors are $C(0, k-1)$ and $C(k, n-1)$

Inductive structure

- Final step combines two subproducts $(M_0 \cdot M_1 \cdots M_{k-1}) \cdot (M_k \cdot M_{k+1} \cdots M_{n-1})$ for some $0 < k < n$
- First factor is $r_0 \times c_{k-1}$, second is $r_k \times c_{n-1}$, where $r_k = c_{k-1}$
- Let $C(0, n{-}1)$ denote the overall cost
- Final multiplication is $O(r_0 r_k c_{n-1})$
- Inductively, costs of factors are $C(0, k{-}1)$ and $C(k, n{-}1)$
- $C(0, n{-}1) =$
  $$C(0, k{-}1) + C(k, n{-}1) + r_0 r_k c_{n-1}$$

Inductive structure

- Final step combines two subproducts $(M_0 \cdot M_1 \cdots M_{k-1}) \cdot (M_k \cdot M_{k+1} \cdots M_{n-1})$ for some $0 < k < n$
- First factor is $r_0 \times c_{k-1}$, second is $r_k \times c_{n-1}$, where $r_k = c_{k-1}$
- Let $C(0, n{-}1)$ denote the overall cost
- Final multiplication is $O(r_0 r_k c_{n-1})$
- Inductively, costs of factors are $C(0, k{-}1)$ and $C(k, n{-}1)$
- $C(0, n{-}1) =$
  $C(0, k{-}1) + C(k, n{-}1) + r_0 r_k c_{n-1}$
- Which k should we choose?

Inductive structure

- Final step combines two subproducts $(M_0 \cdot M_1 \cdots M_{k-1}) \cdot (M_k \cdot M_{k+1} \cdots M_{n-1})$ for some $0 < k < n$
- First factor is $r_0 \times c_{k-1}$, second is $r_k \times c_{n-1}$, where $r_k = c_{k-1}$
- Let $C(0, n{-}1)$ denote the overall cost
- Final multiplication is $O(r_0 r_k c_{n-1})$
- Inductively, costs of factors are $C(0, k{-}1)$ and $C(k, n{-}1)$
- $C(0, n{-}1) =$
  $$C(0, k{-}1) + C(k, n{-}1) + r_0 r_k c_{n-1}$$

- Which k should we choose?
  - Try all and choose the minimum!

Inductive structure

- Final step combines two subproducts $(M_0 \cdot M_1 \cdots M_{k-1}) \cdot (M_k \cdot M_{k+1} \cdots M_{n-1})$ for some $0 < k < n$
- First factor is $r_0 \times c_{k-1}$, second is $r_k \times c_{n-1}$, where $r_k = c_{k-1}$
- Let $C(0, n{-}1)$ denote the overall cost
- Final multiplication is $O(r_0 r_k c_{n-1})$
- Inductively, costs of factors are $C(0, k{-}1)$ and $C(k, n{-}1)$
- $C(0, n{-}1) =$
  $$C(0, k{-}1) + C(k, n{-}1) + r_0 r_k c_{n-1}$$

- Which k should we choose?
  - Try all and choose the minimum!
- Subproblems?

Inductive structure

- Final step combines two subproducts $(M_0 \cdot M_1 \cdots M_{k-1}) \cdot (M_k \cdot M_{k+1} \cdots M_{n-1})$ for some $0 < k < n$
- First factor is $r_0 \times c_{k-1}$, second is $r_k \times c_{n-1}$, where $r_k = c_{k-1}$
- Let $C(0, n{-}1)$ denote the overall cost
- Final multiplication is $O(r_0 r_k c_{n-1})$
- Inductively, costs of factors are $C(0, k{-}1)$ and $C(k, n{-}1)$
- $C(0, n{-}1) =$
  $$C(0, k{-}1) + C(k, n{-}1) + r_0 r_k c_{n-1}$$

- Which k should we choose?
  - Try all and choose the minimum!
- Subproblems?
  - $M_0 \cdot M_1 \cdots M_{k-1}$ would decompose as $(M_0 \cdots M_{j-1}) \cdot (M_j \cdots M_{k-1})$
  - Generic subproblem is $M_j \cdot M_{j+1} \cdots M_k$

Inductive structure

- Final step combines two subproducts $(M_0 \cdot M_1 \cdots M_{k-1}) \cdot (M_k \cdot M_{k+1} \cdots M_{n-1})$ for some $0 < k < n$
- First factor is $r_0 \times c_{k-1}$, second is $r_k \times c_{n-1}$, where $r_k = c_{k-1}$
- Let $C(0, n{-}1)$ denote the overall cost
- Final multiplication is $O(r_0 r_k c_{n-1})$
- Inductively, costs of factors are $C(0, k{-}1)$ and $C(k, n{-}1)$
- $C(0, n{-}1) =$
  $C(0, k{-}1) + C(k, n{-}1) + r_0 r_k c_{n-1}$

- Which k should we choose?
  - Try all and choose the minimum!
- Subproblems?
  - $M_0 \cdot M_1 \cdots M_{k-1}$ would decompose as $(M_0 \cdots M_{j-1}) \cdot (M_j \cdots M_{k-1})$
  - Generic subproblem is $M_j \cdot M_{j+1} \cdots M_k$
- $C(j, k) =$
  $\min_{j<\ell\leq k} [C(j, \ell{-}1) + C(\ell, k) + r_j r_\ell c_k]$

Inductive structure

- Final step combines two subproducts $(M_0 \cdot M_1 \cdots M_{k-1}) \cdot (M_k \cdot M_{k+1} \cdots M_{n-1})$ for some $0 < k < n$
- First factor is $r_0 \times c_{k-1}$, second is $r_k \times c_{n-1}$, where $r_k = c_{k-1}$
- Let $C(0, n{-}1)$ denote the overall cost
- Final multiplication is $O(r_0 r_k c_{n-1})$
- Inductively, costs of factors are $C(0, k{-}1)$ and $C(k, n{-}1)$
- $C(0, n{-}1) =$
  $$C(0, k{-}1) + C(k, n{-}1) + r_0 r_k c_{n-1}$$
- Which k should we choose?
  - Try all and choose the minimum!
- Subproblems?
  - $M_0 \cdot M_1 \cdots M_{k-1}$ would decompose as $(M_0 \cdots M_{j-1}) \cdot (M_j \cdots M_{k-1})$
  - Generic subproblem is $M_j \cdot M_{j+1} \cdots M_k$
- $C(j, k) =$ $\min_{j < \ell \leq k} [C(j, \ell{-}1) + C(\ell, k) + r_j r_\ell c_k]$
- Base case: $C(j, j) = 0$ for $0 \leq j < n$

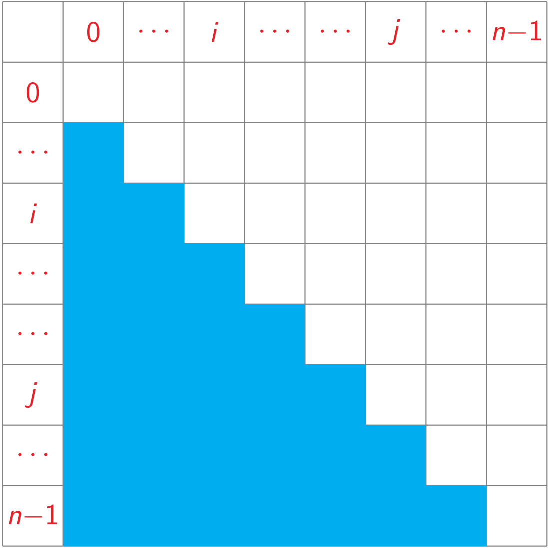
Subproblem dependency

- Compute $C(i,j)$, $0 \leq i,j < n$

	0	$\cdots$	i	$\cdots$	$\cdots$	j	$\cdots$	$n{-}1$
0								
$\cdots$								
i								
$\cdots$								
$\cdots$								
j								
$\cdots$								
$n{-}1$								

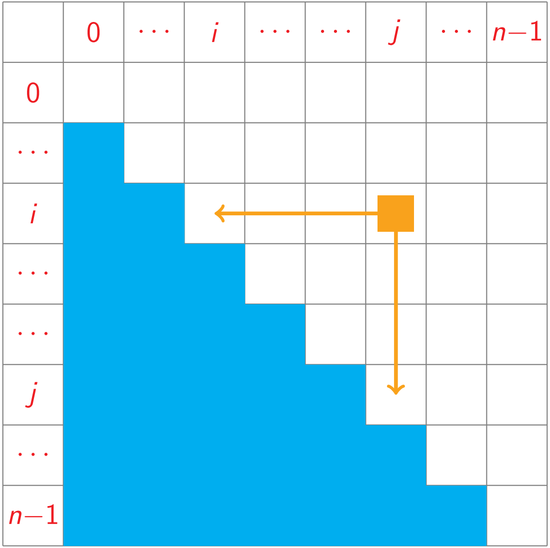
Subproblem dependency

- Compute $C(i,j)$, $0 \leq i,j < n$
  - Only for $i \leq j$
  - Entries above main diagonal



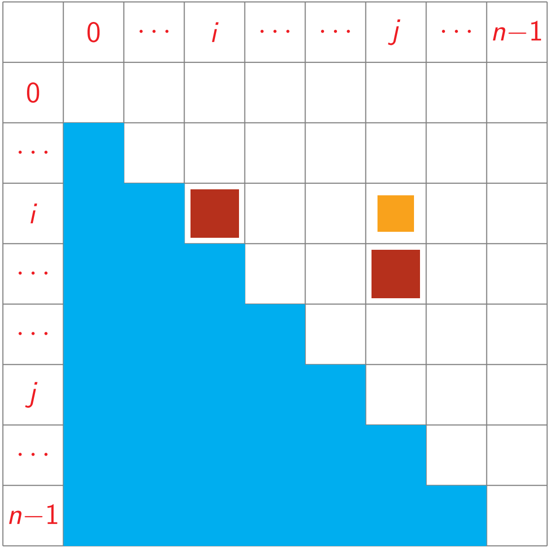
Subproblem dependency

- Compute $C(i,j)$, $0 \leq i,j < n$
  - Only for $i \leq j$
  - Entries above main diagonal
- $C(i,j)$ depends on $C(i,k-1)$, $C(k,j)$ for every $i < k \leq j$



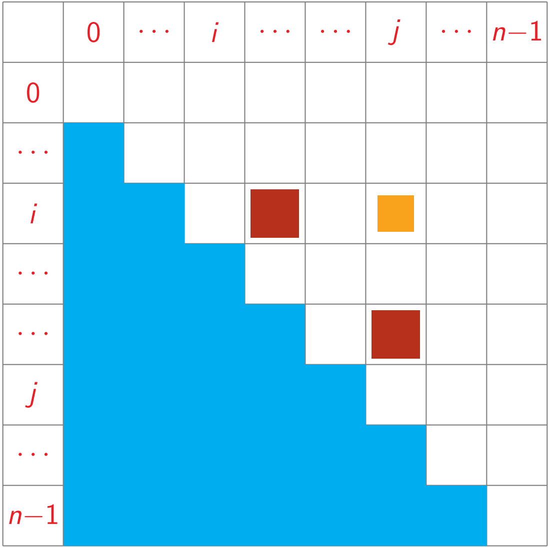
Subproblem dependency

- Compute $C(i,j)$, $0 \leq i,j < n$
  - Only for $i \leq j$
  - Entries above main diagonal
- $C(i,j)$ depends on $C(i,k-1)$, $C(k,j)$ for every $i < k \leq j$



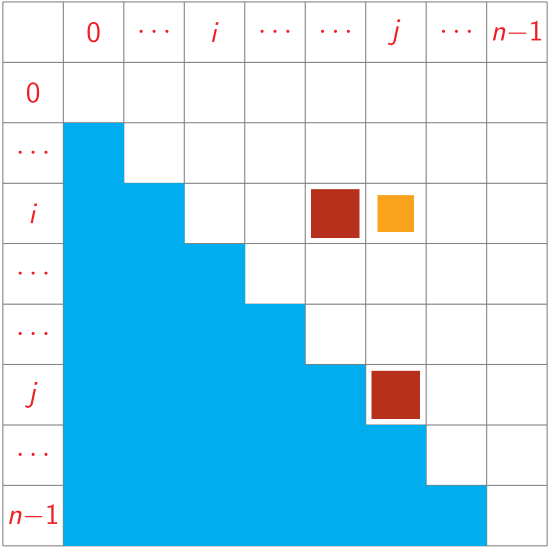
Subproblem dependency

- Compute $C(i,j)$, $0 \leq i,j < n$
  - Only for $i \leq j$
  - Entries above main diagonal
- $C(i,j)$ depends on $C(i,k-1)$, $C(k,j)$ for every $i < k \leq j$



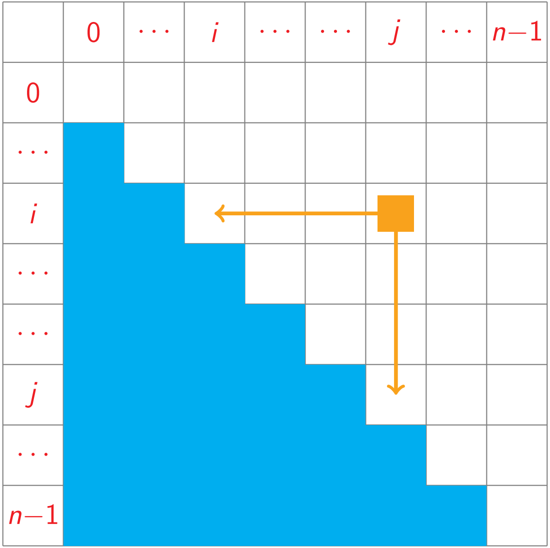
Subproblem dependency

- Compute $C(i, j)$, $0 \leq i, j < n$
  - Only for $i \leq j$
  - Entries above main diagonal
- $C(i, j)$ depends on $C(i, k-1)$, $C(k, j)$ for every $i < k \leq j$



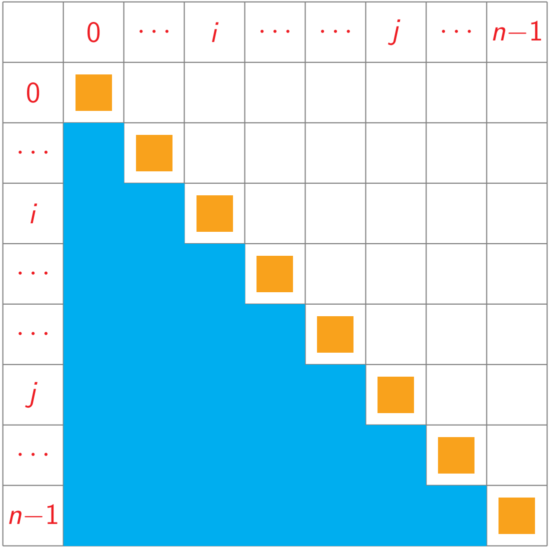
Subproblem dependency

- Compute $C(i,j)$, $0 \leq i,j < n$
  - Only for $i \leq j$
  - Entries above main diagonal
- $C(i,j)$ depends on $C(i,k-1)$, $C(k,j)$ for every $i < k \leq j$
  - $O(n)$ dependencies per entry, unlike LCW, LCS and ED



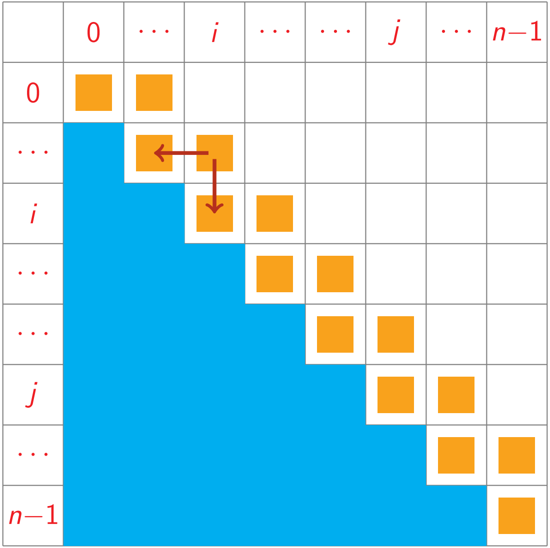
Subproblem dependency

- Compute $C(i,j)$, $0 \leq i,j < n$
  - Only for $i \leq j$
  - Entries above main diagonal
- $C(i,j)$ depends on $C(i,k-1)$, $C(k,j)$ for every $i < k \leq j$
  - $O(n)$ dependencies per entry, unlike LCW, LCS and ED
- Diagonal entries are base case



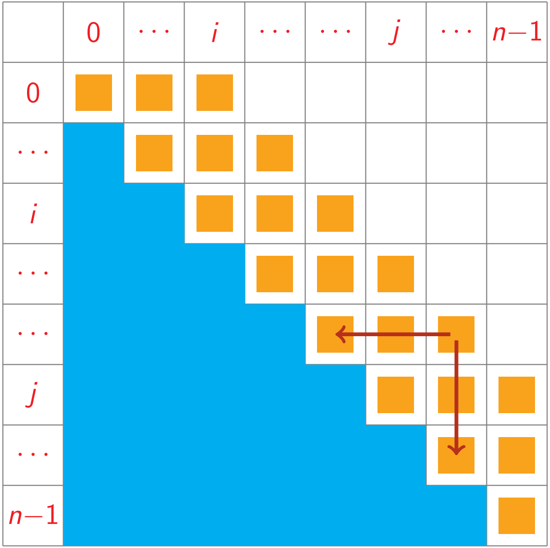
Subproblem dependency

- Compute $C(i,j)$, $0 \leq i,j < n$
  - Only for $i \leq j$
  - Entries above main diagonal
- $C(i,j)$ depends on $C(i,k-1)$, $C(k,j)$ for every $i < k \leq j$
  - $O(n)$ dependencies per entry, unlike LCW, LCS and ED
- Diagonal entries are base case
- Fill matrix by diagonal, from main diagonal



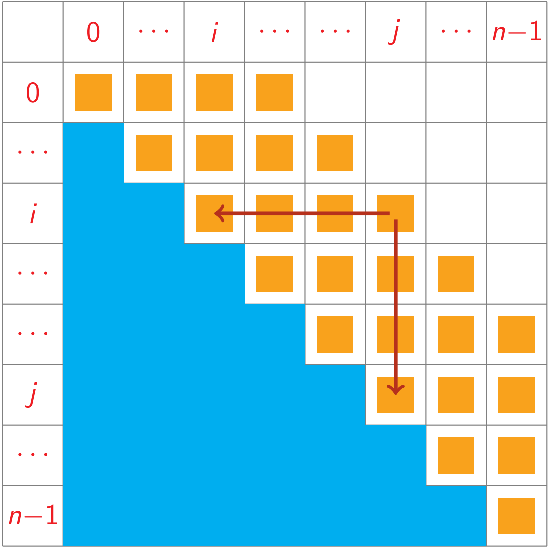
Subproblem dependency

- Compute $C(i,j)$, $0 \leq i,j < n$
  - Only for $i \leq j$
  - Entries above main diagonal
- $C(i,j)$ depends on $C(i,k-1)$, $C(k,j)$ for every $i < k \leq j$
  - $O(n)$ dependencies per entry, unlike LCW, LCS and ED
- Diagonal entries are base case
- Fill matrix by diagonal, from main diagonal



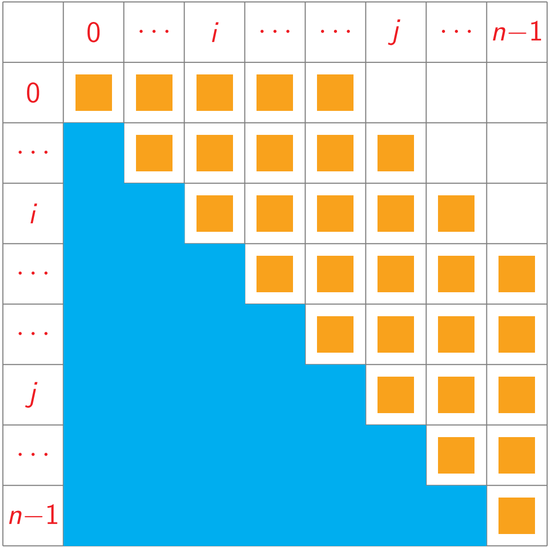
Subproblem dependency

- Compute $C(i,j)$, $0 \leq i,j < n$
  - Only for $i \leq j$
  - Entries above main diagonal
- $C(i,j)$ depends on $C(i,k-1)$, $C(k,j)$ for every $i < k \leq j$
  - $O(n)$ dependencies per entry, unlike LCW, LCS and ED
- Diagonal entries are base case
- Fill matrix by diagonal, from main diagonal



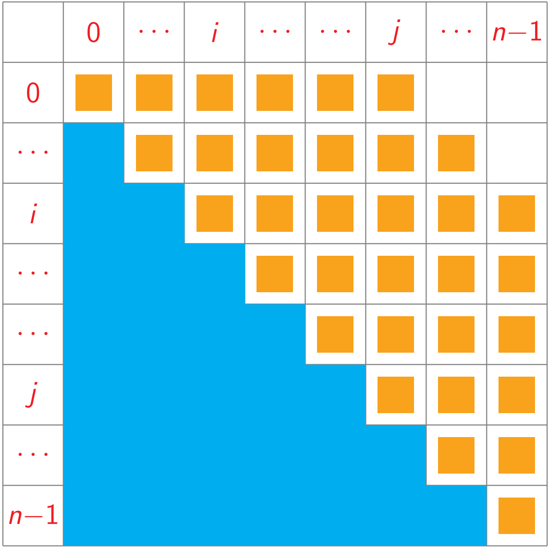
Subproblem dependency

- Compute $C(i,j)$, $0 \leq i,j < n$
  - Only for $i \leq j$
  - Entries above main diagonal
- $C(i,j)$ depends on $C(i,k-1)$, $C(k,j)$ for every $i < k \leq j$
  - $O(n)$ dependencies per entry, unlike LCW, LCS and ED
- Diagonal entries are base case
- Fill matrix by diagonal, from main diagonal



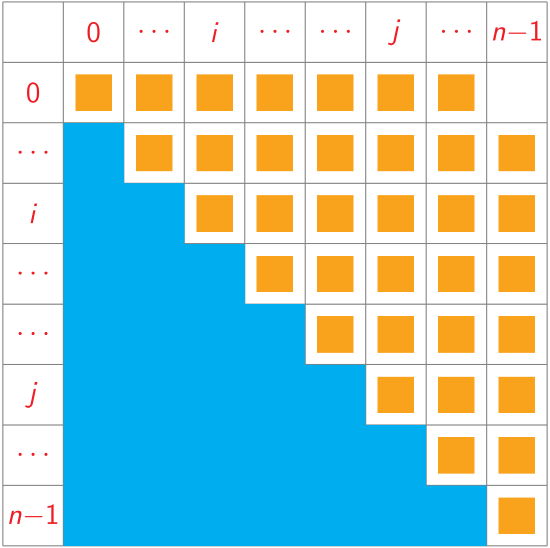
Subproblem dependency

- Compute $C(i,j)$, $0 \le i,j < n$
  - Only for $i \le j$
  - Entries above main diagonal
- $C(i,j)$ depends on $C(i,k{-}1)$, $C(k,j)$ for every $i < k \le j$
  - $O(n)$ dependencies per entry, unlike LCW, LCS and ED
- Diagonal entries are base case
- Fill matrix by diagonal, from main diagonal



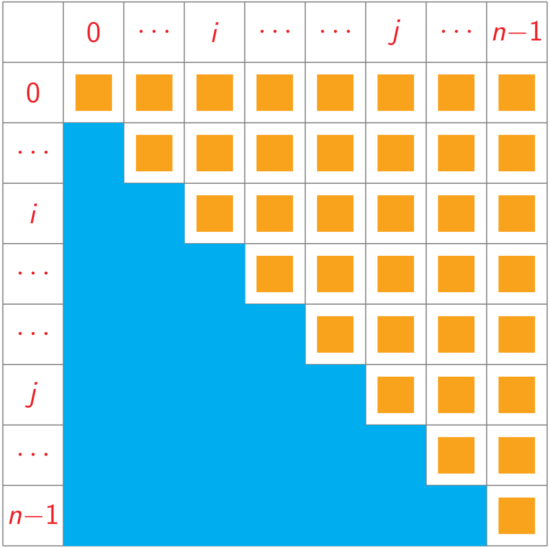
Subproblem dependency

- Compute $C(i,j)$, $0 \leq i,j < n$
  - Only for $i \leq j$
  - Entries above main diagonal
- $C(i,j)$ depends on $C(i,k-1)$, $C(k,j)$ for every $i < k \leq j$
  - $O(n)$ dependencies per entry, unlike LCW, LCS and ED
- Diagonal entries are base case
- Fill matrix by diagonal, from main diagonal



Subproblem dependency

- Compute $C(i,j)$, $0 \leq i,j < n$
  - Only for $i \leq j$
  - Entries above main diagonal
- $C(i,j)$ depends on $C(i,k-1)$, $C(k,j)$ for every $i < k \leq j$
  - $O(n)$ dependencies per entry, unlike LCW, LCS and ED
- Diagonal entries are base case
- Fill matrix by diagonal, from main diagonal



Implementation

```
def C(dim):
  # dim: dimension matrix,
  #      entries are pairs (r_i,c_i)
  import numpy as np
  n = dim.shape[0]
  C = np.zeros((n,n))
  for i in range(n):
    C[i,i] = 0
  for diff in range(1,n):
    for i in range(0,n-diff):
      j = i + diff
      C[i,j] = C[i,i] +
               C[i+1,j] +
               dim[i][0]*dim[i+1][0]*dim[j][1]
      for k in range(i+1,j+1):
        C[i,j] = min(C[i,j],
                     C[i,k-1] + C[k,j] +
                     dim[i][0]*dim[k][0]*dim[j][1])
  return(C[0,n-1])
```

Implementation

```
def C(dim):
  # dim: dimension matrix,
  #      entries are pairs (r_i,c_i)
  import numpy as np
  n = dim.shape[0]
  C = np.zeros((n,n))
  for i in range(n):
    C[i,i] = 0
  for diff in range(1,n):
    for i in range(0,n-diff):
      j = i + diff
      C[i,j] = C[i,i] +
               C[i+1,j] +
               dim[i][0]*dim[i+1][0]*dim[j][1]
      for k in range(i+1,j+1):
        C[i,j] = min(C[i,j],
                     C[i,k-1] + C[k,j] +
                     dim[i][0]*dim[k][0]*dim[j][1])
  return(C[0,n-1])
```

Complexity

Implementation

```
def C(dim):
  # dim: dimension matrix,
  #      entries are pairs (r_i,c_i)
  import numpy as np
  n = dim.shape[0]
  C = np.zeros((n,n))
  for i in range(n):
    C[i,i] = 0
  for diff in range(1,n):
    for i in range(0,n-diff):
      j = i + diff
      C[i,j] = C[i,i] +
               C[i+1,j] +
               dim[i][0]*dim[i+1][0]*dim[j][1]
      for k in range(i+1,j+1):
        C[i,j] = min(C[i,j],
                     C[i,k-1] + C[k,j] +
                     dim[i][0]*dim[k][0]*dim[j][1])
  return(C[0,n-1])
```

Complexity

- We have to fill a table of size $O(n^2)$

Implementation

```
def C(dim):
  # dim: dimension matrix,
  #      entries are pairs (r_i,c_i)
  import numpy as np
  n = dim.shape[0]
  C = np.zeros((n,n))
  for i in range(n):
    C[i,i] = 0
  for diff in range(1,n):
    for i in range(0,n-diff):
      j = i + diff
      C[i,j] = C[i,i] +
               C[i+1,j] +
               dim[i][0]*dim[i+1][0]*dim[j][1]
      for k in range(i+1,j+1):
        C[i,j] = min(C[i,j],
                     C[i,k-1] + C[k,j] +
                     dim[i][0]*dim[k][0]*dim[j][1])
  return(C[0,n-1])
```

Complexity

- We have to fill a table of size $O(n^2)$
- Filling each entry takes $O(n)$

Implementation

```
def C(dim):
  # dim: dimension matrix,
  #      entries are pairs (r_i,c_i)
  import numpy as np
  n = dim.shape[0]
  C = np.zeros((n,n))
  for i in range(n):
    C[i,i] = 0
  for diff in range(1,n):
    for i in range(0,n-diff):
      j = i + diff
      C[i,j] = C[i,i] +
               C[i+1,j] +
               dim[i][0]*dim[i+1][0]*dim[j][1]
      for k in range(i+1,j+1):
        C[i,j] = min(C[i,j],
                     C[i,k-1] + C[k,j] +
                     dim[i][0]*dim[k][0]*dim[j][1])
  return(C[0,n-1])
```

Complexity

- We have to fill a table of size $O(n^2)$
- Filling each entry takes $O(n)$
- Overall, $O(n^3)$