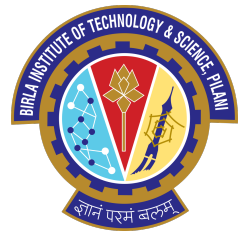
CS F364: Design & Analysis of Algorithms

Lecture-kt12: Augmenting DS (contd.) + Dynamic Programming



Dr. Kamlesh Tiwari,
Assistant Professor,
Department of Computer Science and Information Systems,
BITS Pilani, Rajasthan-333031 INDIA

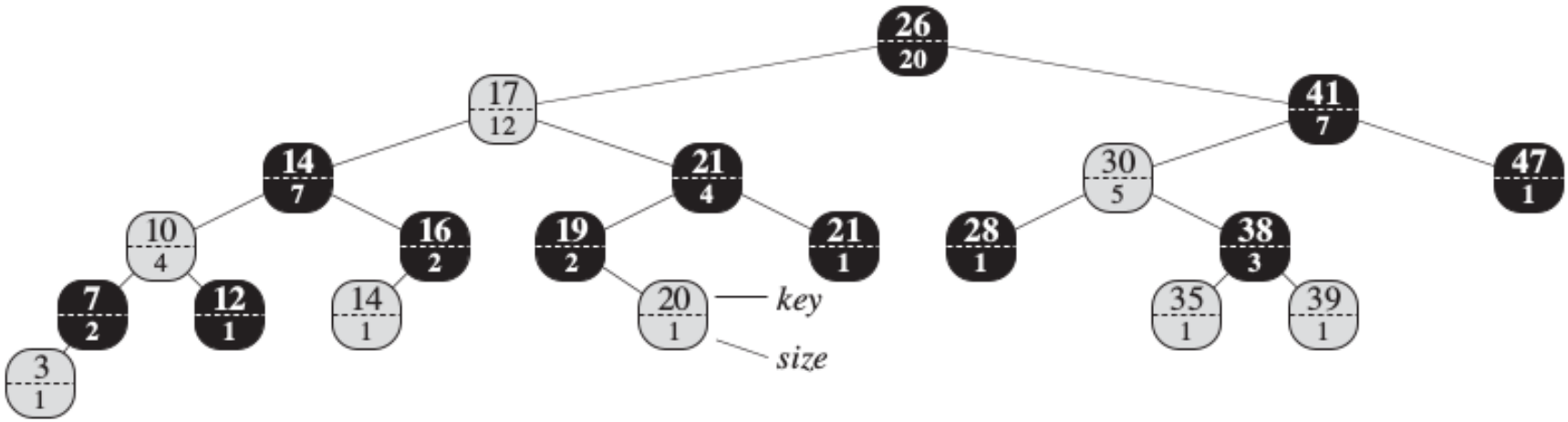
Feb 21, 2017 (Campus @ BITS-Pilani Jan-May 2017)

Recap: Augmenting Data Structure

- **Order statistics tree** maintains a variable called *size* defined as $x.size = x.left.size + x.right.size + 1$

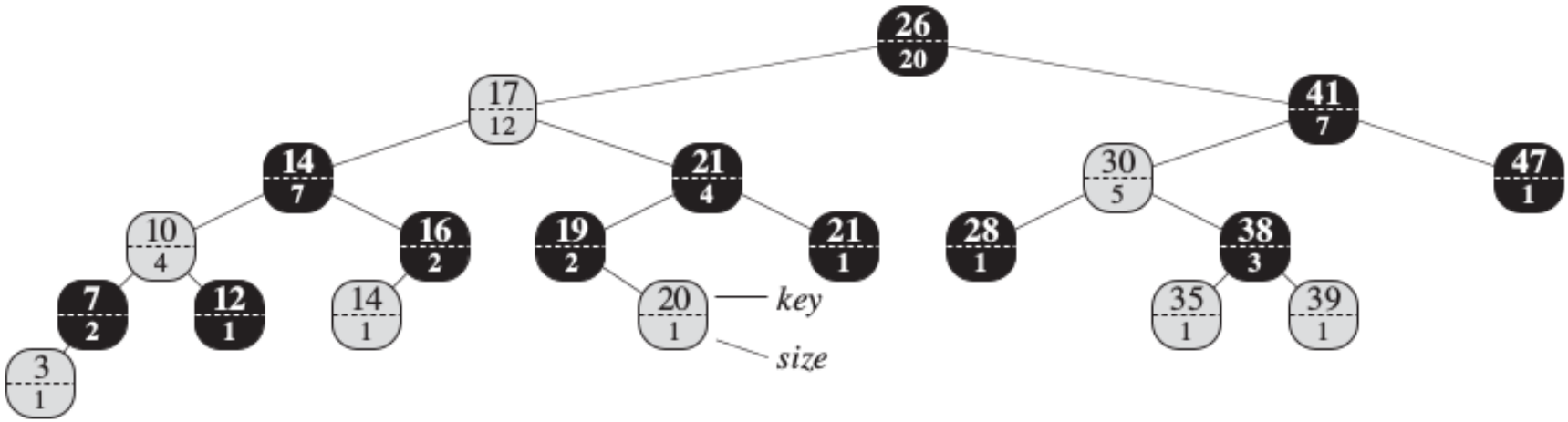
Recap: Augmenting Data Structure

- **Order statistics tree** maintains a variable called *size* defined as $x.size = x.left.size + x.right.size + 1$
- Operations such as 1) determining rank of an item, or 2) selecting an item with given rank can both be performed in $\boxed{O(\log n)}$ time



Recap: Augmenting Data Structure

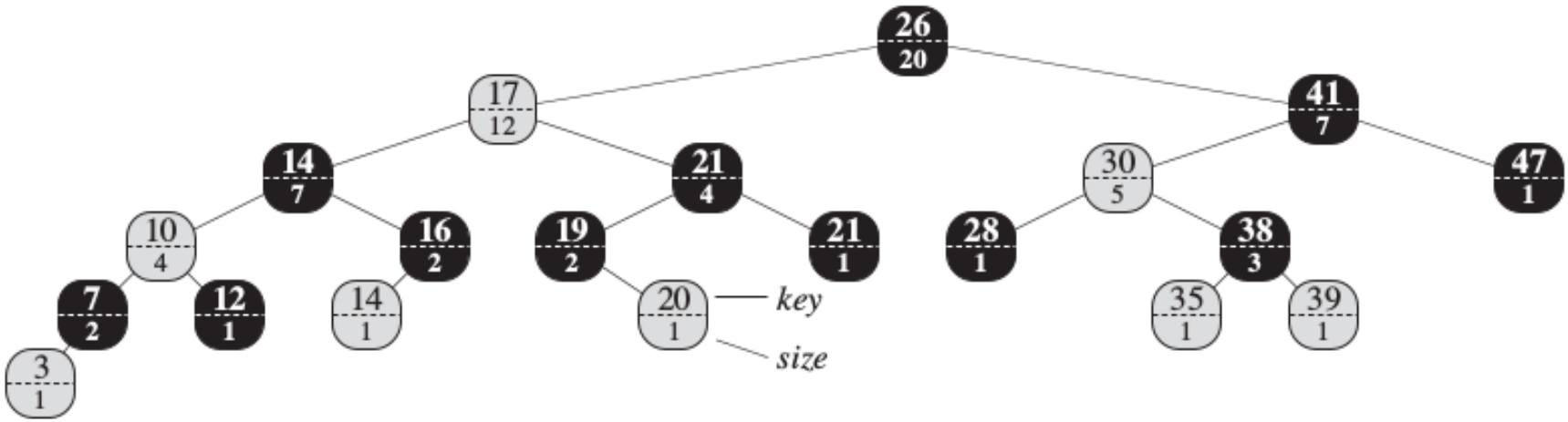
- **Order statistics tree** maintains a variable called *size* defined as $x.size = x.left.size + x.right.size + 1$
- Operations such as 1) determining rank of an item, or 2) selecting an item with given rank can both be performed in $\boxed{O(\log n)}$ time



- Similarly, **interval trees**, wish to query an interval database to find out what events occurred during a given interval.

Recap: Augmenting Data Structure

- **Order statistics tree** maintains a variable called *size* defined as $x.size = x.left.size + x.right.size + 1$
- Operations such as 1) determining rank of an item, or 2) selecting an item with given rank can both be performed in $\boxed{O(\log n)}$ time



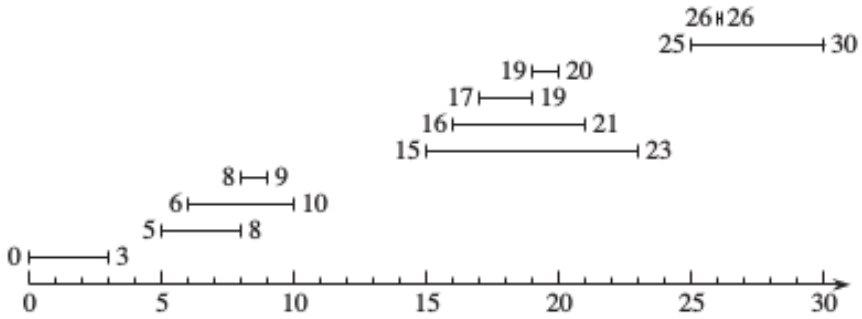
- Similarly, **interval trees**, wish to query an interval database to find out what events occurred during a given interval.
- **Intervals**, i and i' **overlap** if $\boxed{i.low \leq i'.high}$ and $\boxed{i'.low \leq i.high}$

Recap: Augmenting Data Structure

- $x.max = max(x.int.high, x.left.max, x.right.max)$

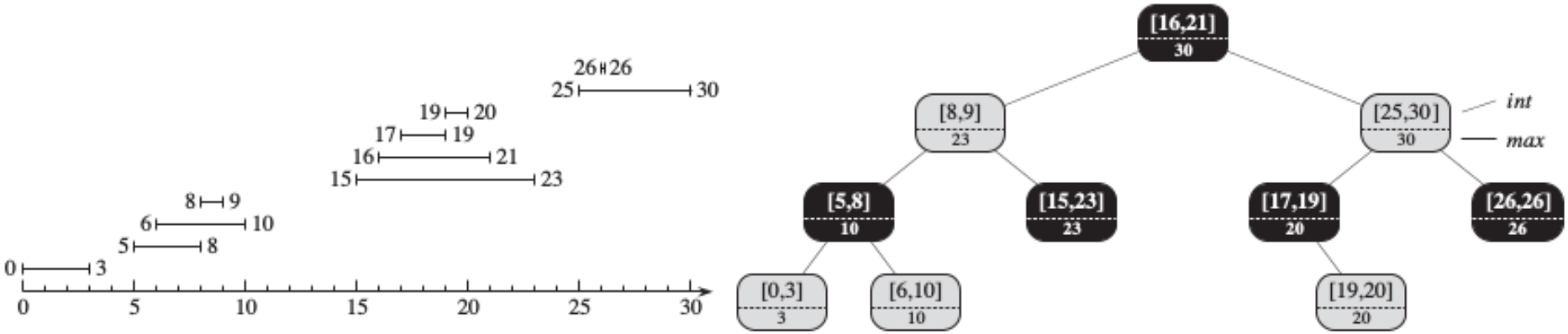
Recap: Augmenting Data Structure

- $x.max = max(x.int.high, x.left.max, x.right.max)$



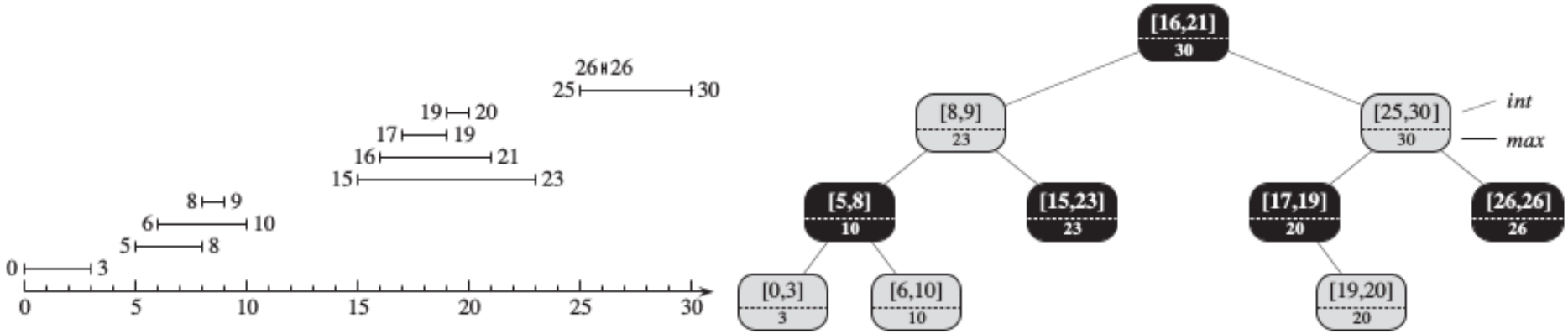
Recap: Augmenting Data Structure

- $x.max = max(x.int.high, x.left.max, x.right.max)$



Recap: Augmenting Data Structure

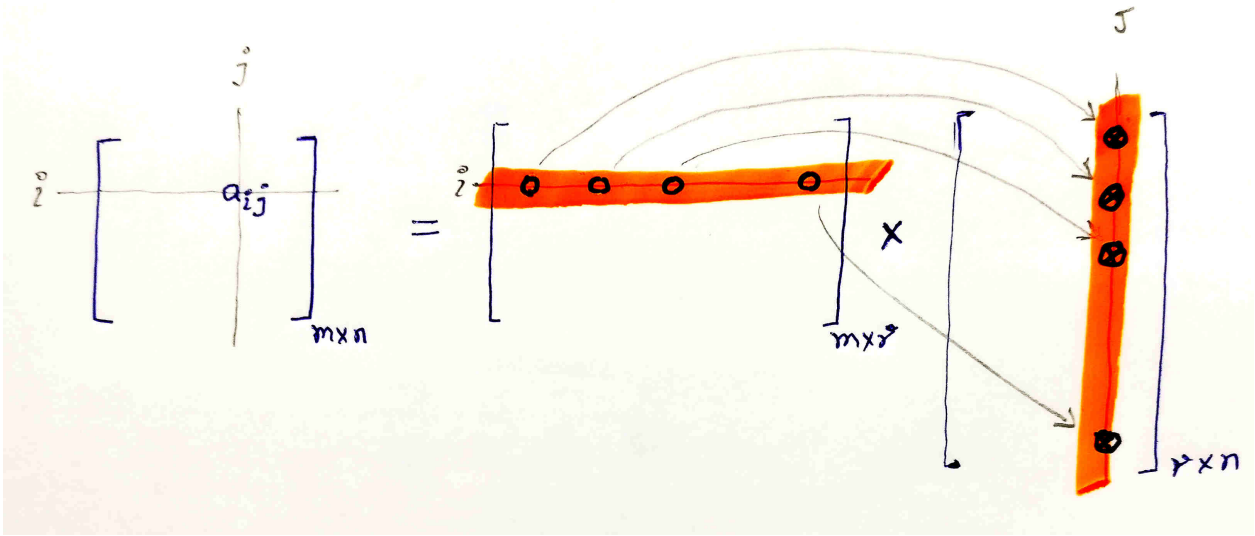
- $x.max = max(x.int.high, x.left.max, x.right.max)$



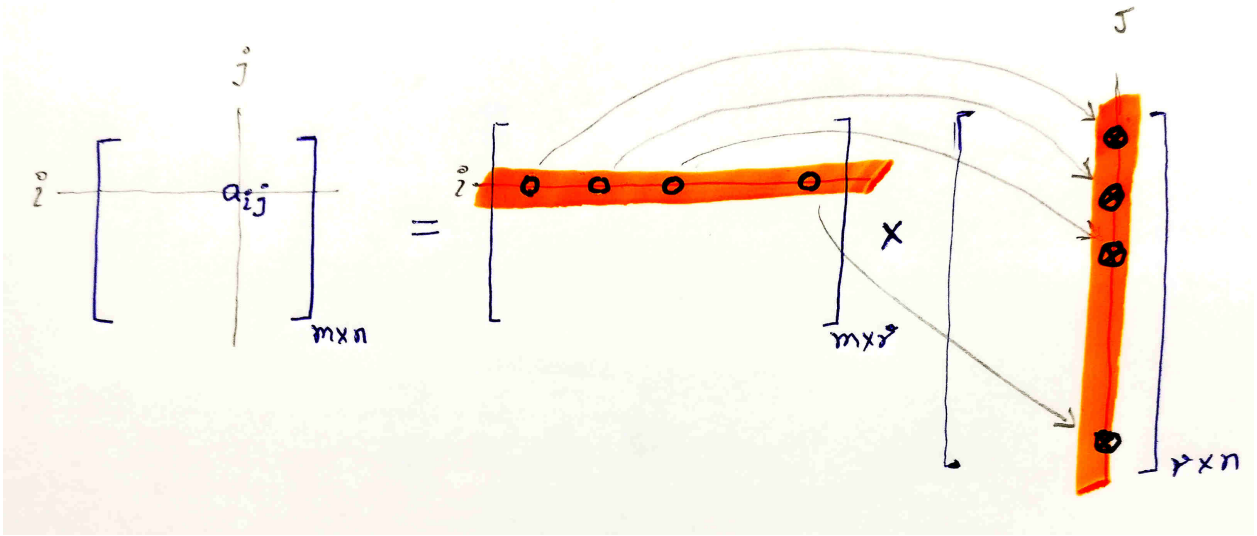
Algorithm 4: Interval-Search(T,i)

```
1 x = T.root
2 while x ≠ T.nil and i does not overlap x.int do
3     if x.left ≠ T.nil and x.left.max ≥ i.low then
4         x = x.left
5     else
6         x = x.right
7 return x
```

Matrix Multiplication

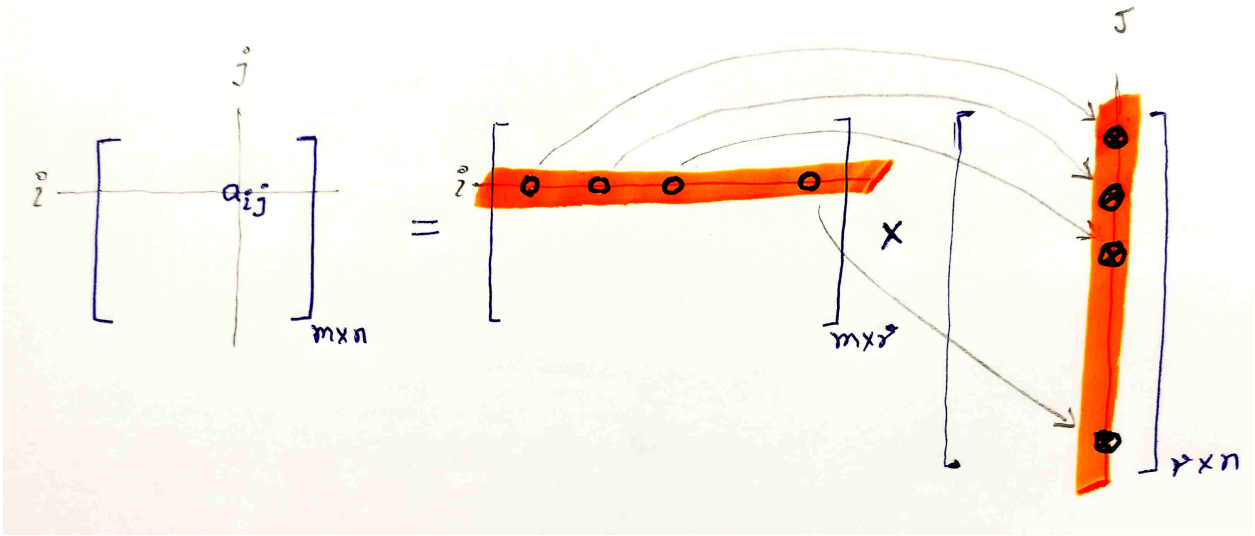


Matrix Multiplication



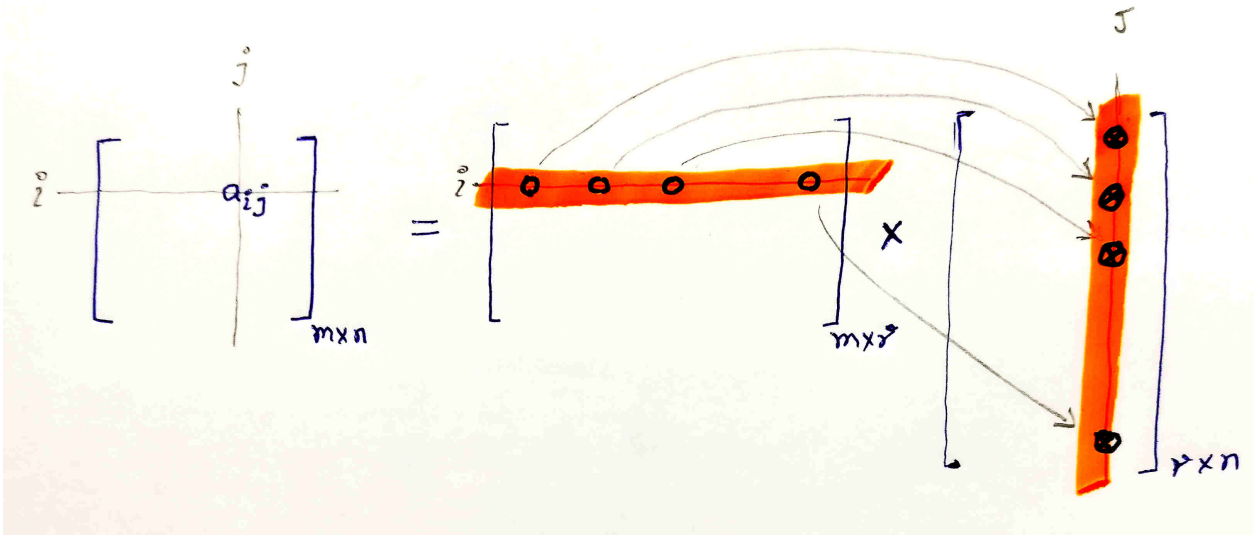
- Matrix multiplication takes $(m \times n) \times r$ steps

Matrix Multiplication



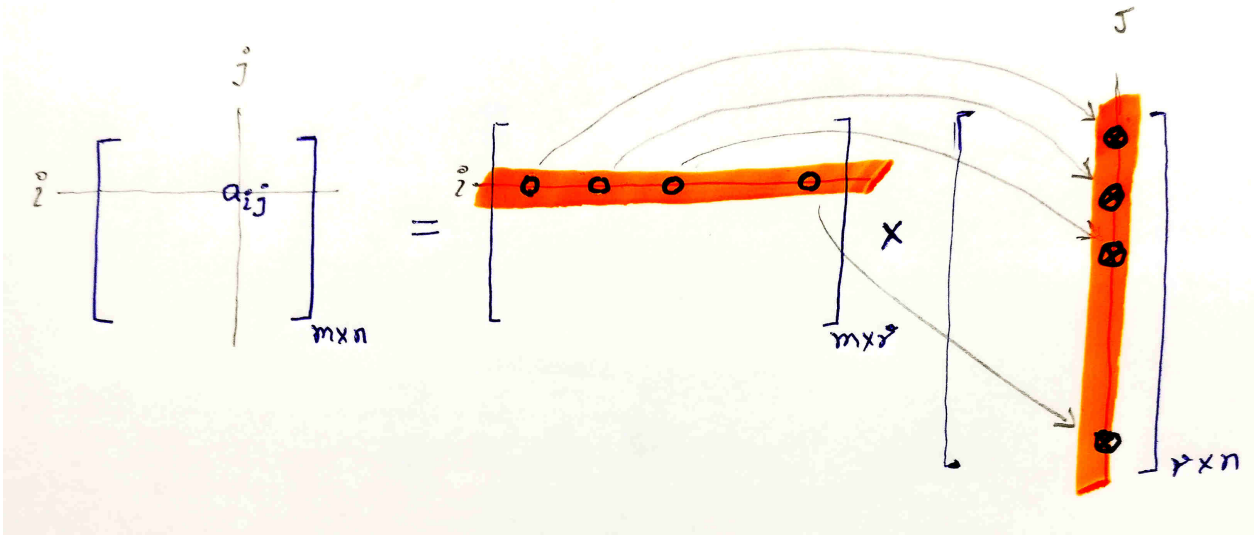
- Matrix multiplication takes $(m \times n) \times r$ steps
- In case you want to multiply three matrices A, B and C of size $u \times v$, $v \times w$ and $w \times z$ respectively,

Matrix Multiplication



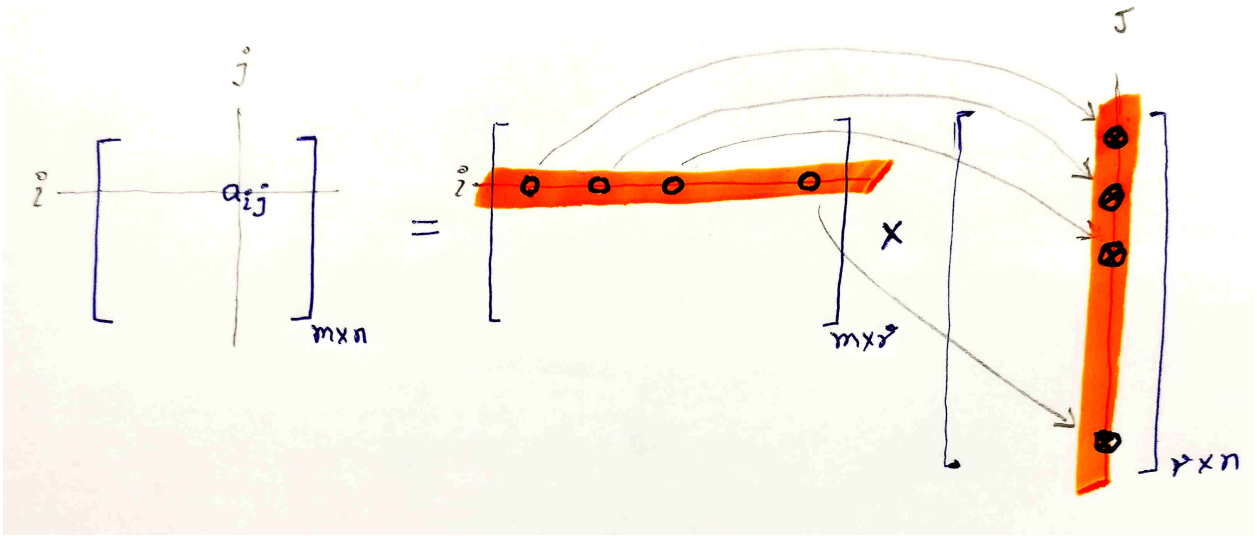
- Matrix multiplication takes $(m \times n) \times r$ steps
- In case you want to multiply three matrices A, B and C of size $u \times v$, $v \times w$ and $w \times z$ respectively, one can do in two ways
  1. $(A \times B) \times C$: takes $u \times v \times w + u \times w \times z$ steps
  2. $A \times (B \times C)$: takes $u \times v \times z + v \times w \times z$ steps

Matrix Multiplication



- Matrix multiplication takes $(m \times n) \times r$ steps
- In case you want to multiply three matrices A, B and C of size $u \times v$, $v \times w$ and $w \times z$ respectively, one can do in two ways
  1. $(A \times B) \times C$: takes $u \times v \times w + u \times w \times z$ steps
  2. $A \times (B \times C)$: takes $u \times v \times z + v \times w \times z$ steps
- If $u = 5, v = 1, w = 3, z = 10$ (in short 5,1,3,10) then it is 165 and 80 respectively

Matrix Multiplication



- Matrix multiplication takes $(m \times n) \times r$ steps
- In case you want to multiply three matrices A, B and C of size $u \times v$, $v \times w$ and $w \times z$ respectively, one can do in two ways
  1. $(A \times B) \times C$: takes $u \times v \times w + u \times w \times z$ steps
  2. $A \times (B \times C)$: takes $u \times v \times z + v \times w \times z$ steps
- If $u = 5, v = 1, w = 3, z = 10$ (in short 5,1,3,10) then it is 165 and 80 respectively

How to determine minimum steps for $v_1, v_2, v_3, \ldots, v_n$

Dynamic Programming (DP)

- DP is like divide-and-conquer and is applied when the subproblems overlap

Dynamic Programming (DP)

- DP is like divide-and-conquer and is applied when the subproblems overlap
- DP solves each subproblem just once and then saves its answer

Dynamic Programming (DP)

- DP is like divide-and-conquer and is applied when the subproblems overlap
- DP solves each subproblem just once and then saves its answer
- Typically applied to optimization problems (can have many possible solutions). And we wish to find an optimal solution

Dynamic Programming (DP)

- DP is like divide-and-conquer and is applied when the subproblems overlap
- DP solves each subproblem just once and then saves its answer
- Typically applied to optimization problems (can have many possible solutions). And we wish to find an optimal solution
- **Elements of dynamic programming**

Dynamic Programming (DP)

- DP is like divide-and-conquer and is applied when the subproblems overlap
- DP solves each subproblem just once and then saves its answer
- Typically applied to optimization problems (can have many possible solutions). And we wish to find an optimal solution
- **Elements of dynamic programming**
  1) Optimal substructure

Dynamic Programming (DP)

- DP is like divide-and-conquer and is applied when the subproblems overlap
- DP solves each subproblem just once and then saves its answer
- Typically applied to optimization problems (can have many possible solutions). And we wish to find an optimal solution
- **Elements of dynamic programming**
  1) Optimal substructure and 2) Overlapping subproblems.

Dynamic Programming (DP)

- DP is like divide-and-conquer and is applied when the subproblems overlap
- DP solves each subproblem just once and then saves its answer
- Typically applied to optimization problems (can have many possible solutions). And we wish to find an optimal solution
- **Elements of dynamic programming**
  1) Optimal substructure and 2) Overlapping subproblems.

DP algorithm needs following four steps

1. Characterize the structure of an optimal solution

Dynamic Programming (DP)

- DP is like divide-and-conquer and is applied when the subproblems overlap
- DP solves each subproblem just once and then saves its answer
- Typically applied to optimization problems (can have many possible solutions). And we wish to find an optimal solution
- **Elements of dynamic programming**
  1) Optimal substructure and 2) Overlapping subproblems.

DP algorithm needs following four steps

1. Characterize the structure of an optimal solution
2. Recursively define the value of an optimal solution

Dynamic Programming (DP)

- DP is like divide-and-conquer and is applied when the subproblems overlap
- DP solves each subproblem just once and then saves its answer
- Typically applied to optimization problems (can have many possible solutions). And we wish to find an optimal solution
- **Elements of dynamic programming**
  1) Optimal substructure and 2) Overlapping subproblems.

DP algorithm needs following four steps

1. Characterize the structure of an optimal solution
2. Recursively define the value of an optimal solution
3. Compute the value of optimal solution in bottom-up fashion

Dynamic Programming (DP)

- DP is like divide-and-conquer and is applied when the subproblems overlap
- DP solves each subproblem just once and then saves its answer
- Typically applied to optimization problems (can have many possible solutions). And we wish to find an optimal solution
- **Elements of dynamic programming**
  1) Optimal substructure and 2) Overlapping subproblems.

DP algorithm needs following four steps

1. Characterize the structure of an optimal solution
2. Recursively define the value of an optimal solution
3. Compute the value of optimal solution in bottom-up fashion
4. Construct an optimal solution from computed information

Matrix-chain multiplication

- Given a sequence (chain) $< A_1, A_2, ..., A_n >$

Matrix-chain multiplication

- Given a sequence (chain) $< A_1, A_2, ..., A_n >$
- Wish to compute product in minimum number of steps.

Matrix-chain multiplication

- Given a sequence (chain) $< A_1, A_2, ..., A_n >$
- Wish to compute product in minimum number of steps.
- **Parenthesize** $< A_1, A_2, A_3, A_4 >$

Matrix-chain multiplication

- Given a sequence (chain) $< A_1, A_2, ..., A_n >$
- Wish to compute product in minimum number of steps.
- **Parenthesize** $< A_1, A_2, A_3, A_4 >$
  1. $A_1(A_2(A_3A_4))$
  2. $A_1((A_2A_3)A_4)$
  3. $(A_1A_2)(A_3A_4)$
  4. $(A_1(A_2A_3))A_4$
  5. $((A_1A_2)A_3)A_4$ There are 5 ways

Matrix-chain multiplication

- Given a sequence (chain) $< A_1, A_2, ..., A_n >$
- Wish to compute product in minimum number of steps.
- **Parenthesize** $< A_1, A_2, A_3, A_4 >$
  1. $A_1(A_2(A_3A_4))$
  2. $A_1((A_2A_3)A_4)$
  3. $(A_1A_2)(A_3A_4)$
  4. $(A_1(A_2A_3))A_4$
  5. $((A_1A_2)A_3)A_4$ There are 5 ways
- Let $P(n)$ be the number of ways to parenthesize n matrices then

$$P(n) = \begin{cases} 1 & \text{if } n = 1 \\ \sum_{k=1}^{n-1} P(k) \times P(n-k) & \text{otherwise} \end{cases}$$

Matrix-chain multiplication

- Given a sequence (chain) $< A_1, A_2, ..., A_n >$
- Wish to compute product in minimum number of steps.
- **Parenthesize** $< A_1, A_2, A_3, A_4 >$
  1. $A_1(A_2(A_3A_4))$
  2. $A_1((A_2A_3)A_4)$
  3. $(A_1A_2)(A_3A_4)$
  4. $(A_1(A_2A_3))A_4$
  5. $((A_1A_2)A_3)A_4$ There are 5 ways
- Let $P(n)$ be the number of ways to parenthesize n matrices then

$$P(n) = \begin{cases} 1 & \text{if } n = 1 \\ \sum_{k=1}^{n-1} P(k) \times P(n-k) & \text{otherwise} \end{cases}$$

- Its solution is Catalan number $P(n) = C(n-1)$, where $C(n) = \frac{1}{n+1}^{2n}C_n = \Omega(4^n/n^{3/2})$.

Matrix-chain multiplication

- Let matrix A_i has dimension $p_{i-1} \times p_i$ and the sequence $< p_0, p_1, ..., p_n >$ represents all input matrices dimensions

Matrix-chain multiplication

- Let matrix A_i has dimension $p_{i-1} \times p_i$ and the sequence $< p_0, p_1, ..., p_n >$ represents all input matrices dimensions
- DP maintains two matrices m and s as below

Matrix-chain multiplication

- Let matrix A_i has dimension $p_{i-1} \times p_i$ and the sequence $< p_0, p_1, ..., p_n >$ represents all input matrices dimensions
- DP maintains two matrices m and s as below

Algorithm 7: Matrix-Chain-Multiply(A, s, i, j)

```
1 if j > i then
2     X = Matrix-Chain-Multiply( A, s, i, s[i][j])
3     Y = Matrix-Chain-Multiply( A, s, s[i][j]+1, j)
4     return X×Y
5 else
6     return A_i
```

Matrix-chain multiplication

Algorithm 8: Matrix-Chain-Order(p)

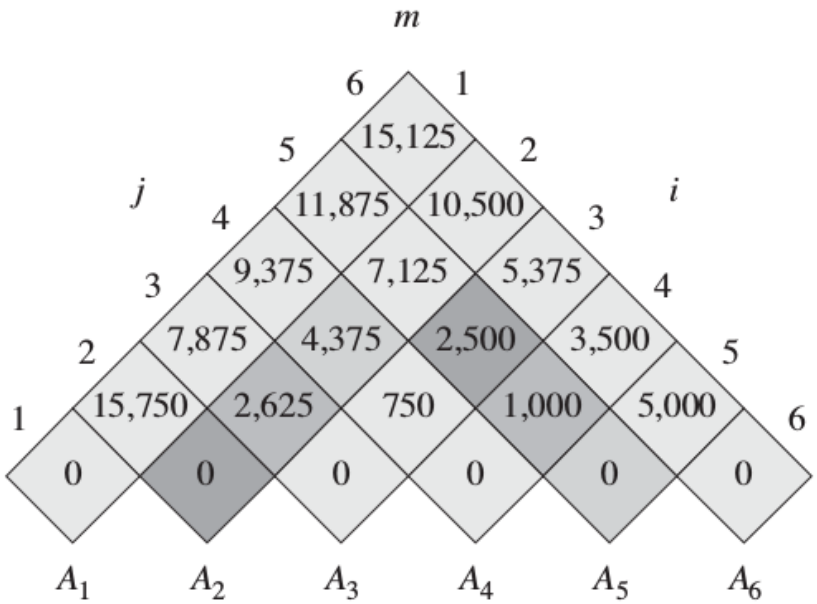
```
1  n = length[p] -1
2  for i = 1 to n do
3      m[i][i] = 0
4  for l = 2 to n do
5      for i = 1 to n-l+1 do
6          j = i + l - 1
7          m[i][j] = ∞
8          for k = i to j-1 do
9              q = m[i][k] + m [k+1][j] + p_{i-1} p_k p_j
10             if q < m[i][j] then
11                 m[i][j] = q
12                 s[i][j] = k
```

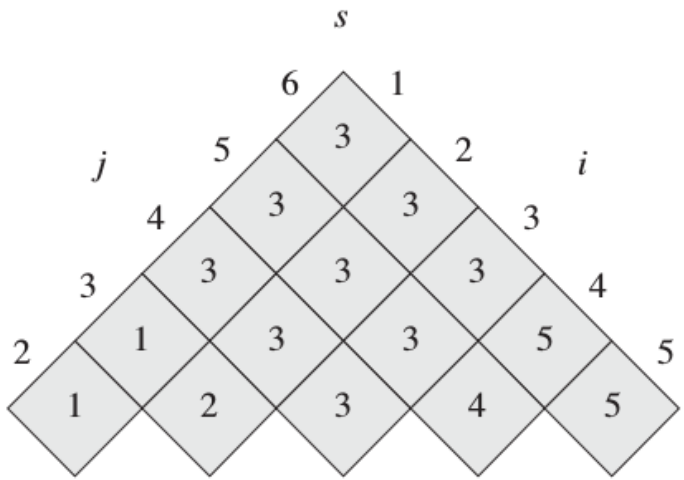
Matrix-chain multiplication

- Consider $p =< 30, 35, 15, 5, 10, 20, 25 >$

Matrix-chain multiplication

- Consider $p = < 30, 35, 15, 5, 10, 20, 25 >$





Longest common subsequence

- A subsequence of a sequence can be obtained by removing zero or more elements.

Longest common subsequence

- A subsequence of a sequence can be obtained by removing zero or more elements.
- In the longest-common-subsequence problem, we are given two sequences $X =< x_1 x_2 x_3 ... x_m >$ and $Y =< y_1 y_2 y_3 ... y_n >$ and wish to find a maximum-length common subsequence of X and Y.

Thank You!

Thank you very much for your attention!

Queries ?