HUST

ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

APPLIED ALGORITHMS

APPLIED ALGORITHMS

Data structures and advanced techniques: Range Minimum Query, Segment Trees

CONTENTS

- Range Minimum Query (cấu trúc truy vấn phần tử nhỏ nhất trên đoạn con)
- Segment tree (cấu trúc cây phân đoạn)

Range Minimum Query (RMQ)

- **Illustrative exercise (P.02.03.01).** Given the sequence $a_0, a_1, \ldots, a_{N-1}$ and a positive integer K. We need to perform K queries, each query of the form **RMQ**(i, j) returns the index of the smallest element of the sequence $a_i, a_{i+1}, \ldots, a_j$.

Range Minimum Query (RMQ)

- **Illustrative exercise (P.02.03.01).** Given the sequence $a_0, a_1, \ldots, a_{N-1}$ and a positive integer K. We need to perform K queries, each query of the form **RMQ**(i, j) returns the index of the smallest element of the sequence $a_i, a_{i+1}, \ldots, a_j$.
- Direct algorithm
  - For each query RMQ(i, j): traverse sequence $a_i, a_{i+1}, \ldots, a_j$.
  - Complexity O(j - i)

```
RMQ(a, i, j){
  min = +∞; min_idx = -1;
  for k = i to j do {
    if min > a[k] then  {
       min = a[k];, min_idx = k;
    }
  }
  return min_idx;
}
```

- **Illustrative exercise (P.02.03.01).** Given the sequence $a_0, a_1, \ldots, a_{N-1}$ and a positive integer K. We need to perform K queries, each query of the form **RMQ**(i, j) returns the index of the smallest element of the sequence $a_i, a_{i+1}, \ldots, a_j$.
- Preprocess
  - Calculate $M[i,j]$ as the index of the smallest element of the subsequence starting from a_j and having 2^i elements, where $i = 0, 1, 2, \ldots, \log_2(N+1)$ and $j = 0, 1, 2, \ldots, N-1$.

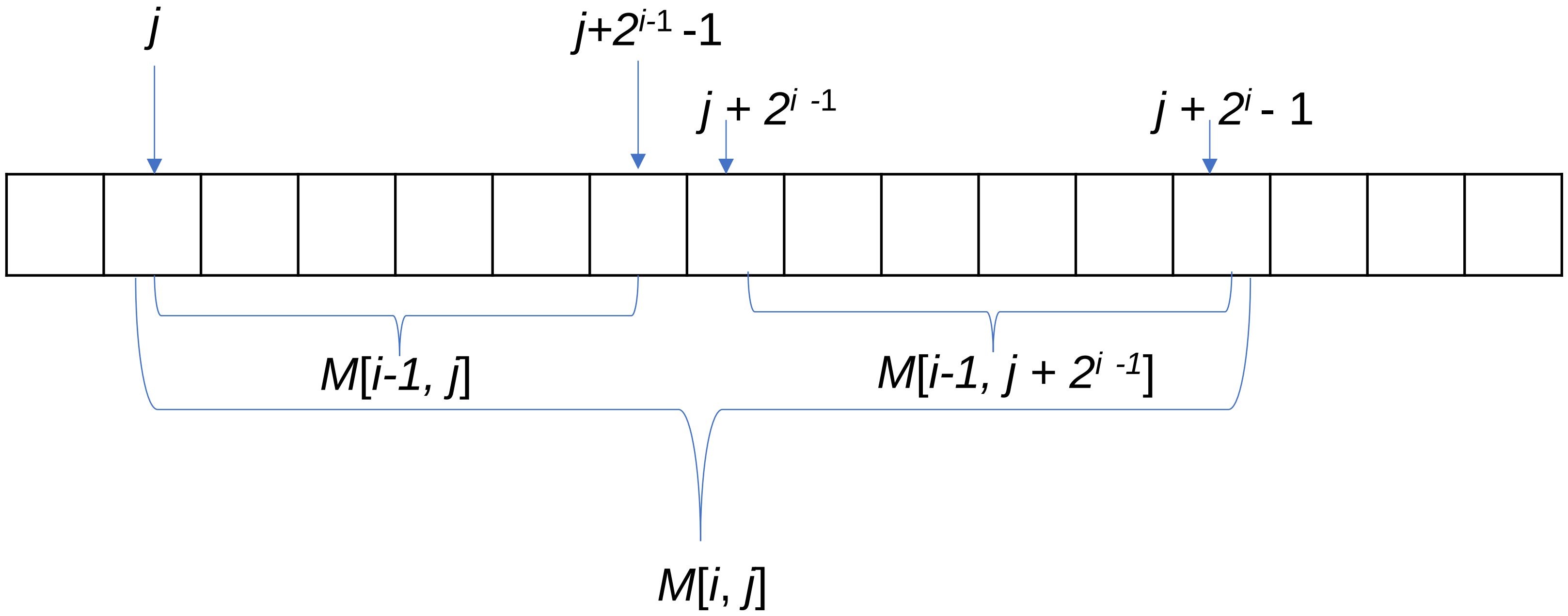
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	**4**	**6**	**1**	**6**	**8**	**7**	**3**	**3**	**5**	**8**	**9**	**1**	**2**	**6**	**4**

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
1	0	1	3	3	4	6	7	8	8	9	10	12	12	13	15	-
2	3	3	3	3	7	8	8	8	8	12	12	12	12	-	-	-
3	3	3	3	3	8	12	12	12	12	-	-	-	-	-	-	-
4	12	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Range Minimum Query (RMQ)

- The smallest subproblem: M[0, j] = j, j = 0,…, N-1
- Recursive formula

$$M[i, j] = \begin{cases} M[i-1, j] \text{ if } a[M[i-1, j]] < a[M[i-1, j+2^{i-1}]] \\ M[i-1, j+2^{i-1}], \text{ otherwise} \end{cases}$$



Range Minimum Query (RMQ)

- The smallest subproblem: $M[0, j] = j$, $j = 0,\dots, N-1$
- Recursive formula

$$M[i, j] = \begin{cases} M[i-1, j] \text{ if } a[M[i-1, j]] < a[M[i-1, j+2^{i-1}]] \\ M[i-1, j+2^{i-1}], \text{ otherwise} \end{cases}$$

```
preprocessing(){
  for (i = 0; i < N; i++) M[0,i] = i;

  for (j = 0; 2^j ≤ N; j++){
    for(i = 0; i + 2^j -1 < N; i++){
      if a[M[j-1,i]] < a[M[j-1,i+2^(j-1)]] then{
        M[j,i] = M[j-1,i];
      }else{
        M[j,i] = M[j-1,i+2^(j-1)];
      }
    }
  }
}
```

Range Minimum Query (RMQ)

- Query **RMQ**(i,j)
  - k = [log(j-i+1)]
  - **RMQ**(i,j) = $M[k,i]$ if $a[M[k,i]] \le a[M[k, j-2^k+1]]$
    $M[k, j-2^k+1]$, otherwise
- **RMQ**(4,14) = ?
  - k = [log(14-4+1)]=3
  - a[7] > a[12] → **RMQ**(4,14) = 12

M[3,7] = 12

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	4	6	1	6	8	7	3	3	5	8	9	1	2	6	4

M[3,4] = 7

SEGMENT TREES (CÂY PHÂN ĐOẠN)

- **Illustrative exercise (P.02.03.02).** Given the sequence $a_1, a_2, \ldots, a_n$. Perform a series of the following operations on the given sequence:
  - update *i* *v*: assign $a_i = v$
  - get-max *i* *j*: return the largest value in the sequence $a_i, a_{i+1}, \ldots, a_j$.

SEGMENT TREES (CÂY PHÂN ĐOẠN)

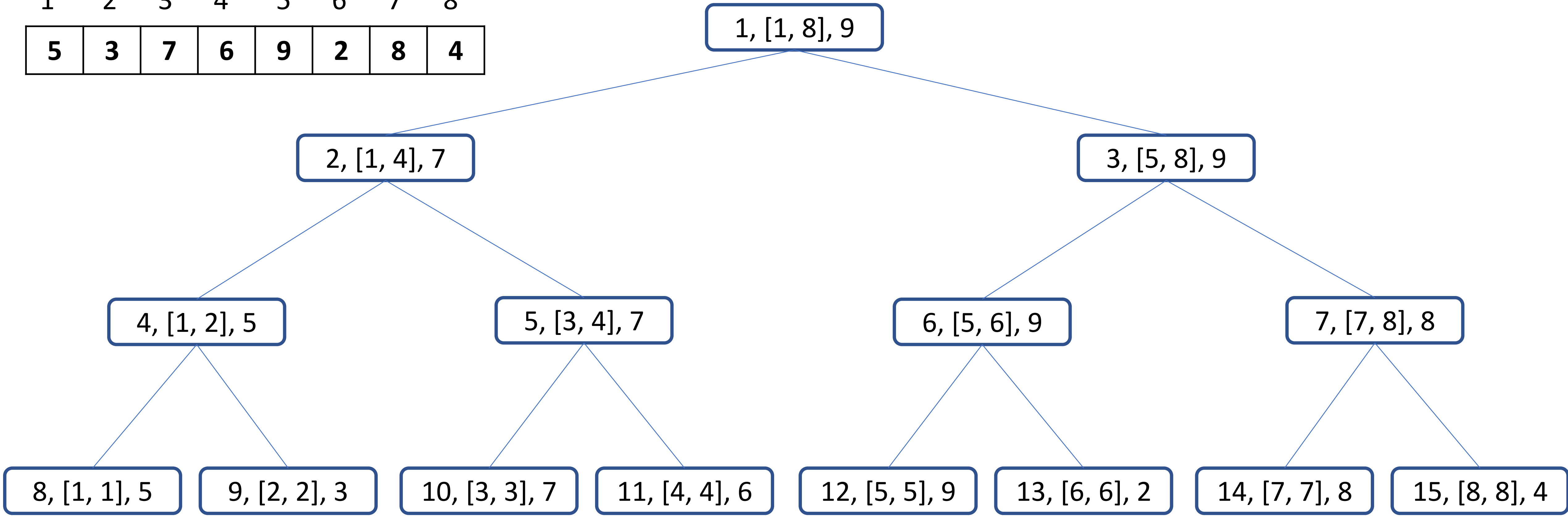
- **Illustrative exercise (P.02.03.02).** Given the sequence $a_1, a_2, \ldots, a_n$. Perform a series of the following operations on the given sequence:
  - update *i* *v*: assign $a_i = v$
  - get-max *i* *j*: return the largest value in the sequence $a_i, a_{i+1}, \ldots, a_j$.
- Direct algorithm
  - Operation update *i* *v*: upadate $a_i = v$, complexity O(1)
  - Operation get-max *i* *j*: traverse sequence $a_i, a_{i+1}, \ldots, a_j$ to find the largest element, complexity $O(j - i)$

SEGMENT TREES (CÂY PHÂN ĐOẠN)

- Segment Trees: full binary tree (cấu trúc cây nhị phân đầy đủ)
  - Each node manages a segment in the tree
  - Root node with *id* = 1 manages the segment with index [1, *n*]
  - Each node with *id* = *v* manages the segment with index [*i*, *j*], then
    - Left child node with *id* = 2*v* manages the segment with index [*i*, (*i*+*j*)/2]
    - Right child node with *id* = 2*v*+1 manages the segment with index [(*i*+*j*)/2+1, *j*]
- Data structure represents each node of the tree → *id*, [*L*, *R*], *maxVal*[*id*]
  - *id*: index of the node
  - *L* and *R*: start index and end index of subsequence $a_L, a_{L+1}, \ldots, a_R$ that the node manages
  - *maxVal*[*id*]: the largest value of subsequence $a_L, a_{L+1}, \ldots, a_R$ that the node manages

SEGMENT TREES (CÂY PHÂN ĐOẠN)

1	2	3	4	5	6	7	8
5	**3**	**7**	**6**	**9**	**2**	**8**	**4**



SEGMENT TREES (CÂY PHÂN ĐOẠN)

```
GetMaxFromNode(id, L, R, i, j){
   // return the max value of ai,. . ., aj from the node (id, L, R)
   if i > R or j < L then return -∞; // [L, R] and [i, j] are disjoint → not found
   if i <= L and j >= R then // [L, R] is within [i, j]
      return maxVal[id] // max value is stored in the node (id, L, R)
   m = (L + R)/2;
   LC = 2*id; RC = 2*id+1; // left-child and right-child
   maxLeft = GetMaxFromNode(LC, L, m, i, j);
   maxRight = GetMaxFromNode(RC, m+1, R, i, j);
   return max(maxLeft, maxRight);
}
GetMax(i, j){
   return GetMaxFromNode(1, 1, n, i, j) // Find Max from the root node
}
```

SEGMENT TREES (CÂY PHÂN ĐOẠN)

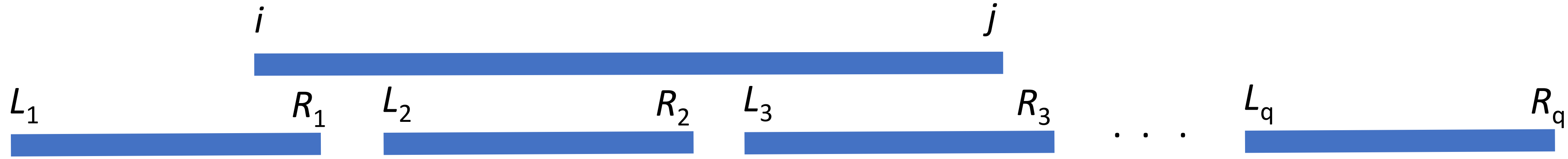
```
UpdateFromNode(id, L, R, index, value){
   // propagate from the node (id, L, R) by the update: a[index] = value
   if L > R then return;
   if index < L or index > R then return; // node (id, L, R) does not manage a[index]
   if L == R then {     maxVal[id] = value; return;  }
   LC = 2*id; RC = 2*id + 1; // left-child and right-child
   m = (L+R)/2;
   UpdateFromNode(LC, L, m, index, value);
   UpdateFromNode(RC, m+1, R, index, value);
   maxVal[id] = max(maxVal[LC], maxVal[RC]);
}
Update(i, v){
   UpdateFromNode(1, 1, n, i, v) // start the propagation from the root node
}
```

SEGMENT TREES (CÂY PHÂN ĐOẠN)

- The number of nodes on the segment tree is less than or equal to 4n
  - Denote $k = \lceil \log n \rceil$
  - The maximum number of nodes in the tree is $1 + 2^1 + 2^2 + \ldots + 2^k = 2^{k+1} - 1 < 4n$

SEGMENT TREES (CÂY PHÂN ĐOẠN)

- The complexity of the GetMax operation: we will traverse at most 4 nodes on each level of the tree (prove by induction)
  - At level 1 (at root node): we only traverse the root node
  - Suppose we are at a current level k, we visit nodes V_k ($|V_k| \leq 4$)
    - We call two segments [a, b] and [c, d] **over-lap** with each other if this segment is not a subsegment (or equal) of the other segment.
    - Note: In the function GetMaxFromNode(*id*, *L*, *R*, *i*, *j*) at node (*id*, *L*, *R*), we only have recursive call to traverse the child node if the segments [*L*, *R*] và [*i*, *j*] are over- lap.
    - Suppose the nodes in V_k (from left to right) are (id_1, L_1, R_1), (id_2, L_2, R_2), ..., (id_q, L_q, R_q). Obviously, the number of segments in [L_1, R_1], . . ., [L_q, R_q] over-lap with segment [*i*, *j*] must be $\leq$ 2 because otherwise, the middle segments in this series of over-lap segments with [*i*, *j*] will definitely is a child segment (or equal) of segment [*i*, *j*] and therefore from the nodes corresponding to the segments in the middle there will be no recursive call to visit the child node. Thus, it can be deduced that the number of child nodes at level k+1 traversed is less than or equal to 4.



SEGMENT TREES (CÂY PHÂN ĐOẠN)

- The complexity of the GetMax operation
  - We will traverse at most 4 nodes on each level of the tree (prove by induction).
  - The height of the tree is O(log*n*). Thus, the complexity of the GetMax(i, j) operation is 4 x O(log*N*) or O(log*N*)

SEGMENT TREES (CÂY PHÂN ĐOẠN)

- The complexity of the Update(*i*, *v*) operation
  - Starting from the root node, at each level *k*, we only visit at most one child node because the index *i* only belongs to at most 1 subsegment among the 2 subsegments divided from the segment at the previous level.
  - Therefore, the complexity of the Update(i, v) operation is the height of the tree and is O(log*n*)

THANK YOU !

HUST

hust.edu.vn fb.com/dhbkhn