ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

THUẬT TOÁN ỨNG DỤNG

CẤU TRÚC DỮ LIỆU VÀ KỸ THUẬT NÂNG CAO

Range Minimum Query, Segment Trees

ONE LOVE. ONE FUTURE.

NỘI DUNG

- Cấu trúc truy vấn phần tử nhỏ nhất trên đoạn con
- Cấu trúc cây phân đoạn

- **Bài tập minh họa (P.02.03.01)**. Cho dãy $a_0, a_1, \ldots, a_{N-1}$. Cho số nguyên dương K, ta cần thực hiện K truy vấn, mỗi truy vấn dạng **RMQ**(i, j) trả về chỉ số của phần tử nhỏ nhất của dãy $a_i, a_{i+1}, \ldots, a_j$.

CẤU TRÚC TRUY VẤN PHẦN TỬ NHỎ NHẤT TRÊN ĐOẠN CON (RMQ)

- **Bài tập minh họa (P.02.03.01)**. Cho dãy a_0, a_1, ..., a_{N-1}. Cho số nguyên dương K, ta cần thực hiện K truy vấn, mỗi truy vấn dạng **RMQ**(i, j) trả về chỉ số của phần tử nhỏ nhất của dãy a_i, a_{i+1}, . . ., a_j.
- Thuật toán trực tiếp
  - Với mỗi truy vẫn RMQ(i, j), ta duyệt dãy a_i, a_{i+1}, . . ., a_j.
  - Độ phức tạp 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;
}
```

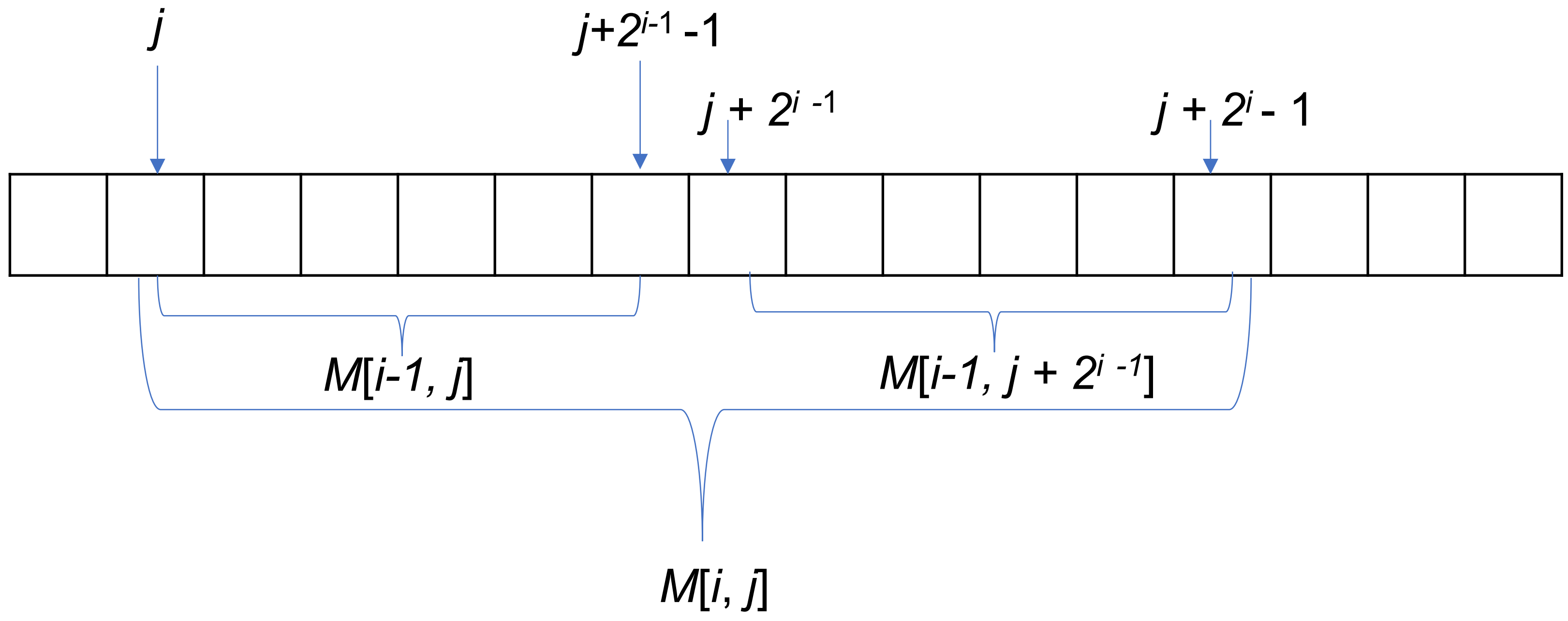
- **Bài tập minh họa (P.02.03.01)**. Cho dãy $a_0, a_1, \ldots, a_{N-1}$. Cho số nguyên dương K, ta cần thực hiện K truy vấn, mỗi truy vấn dạng **RMQ**(i, j) trả về chỉ số của phần tử nhỏ nhất của dãy $a_i, a_{i+1}, \ldots, a_j$.
- Tiền xử lý
  - Tính $M[i,j]$ là chỉ số của phần tử nhỏ nhất của dãy con bắt đầu từ a_j và có 2^i phần tử, với i = 0, 1, 2, ..., $\log_2(N+1)$ và j = 0, 1, 2, ..., 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	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

CẤU TRÚC TRUY VẤN PHẦN TỬ NHỎ NHẤT TRÊN ĐOẠN CON (RMQ)

- Bài toán con nhỏ nhất $M[0, j] = j$, $j = 0,\ldots, N-1$
- Công thức truy hồi
- $M[i, j] = \begin{cases} M[i-1, j] \text{ nếu } a[M[i-1, j]] < a[M[i-1, j+2^{i-1}]] \\ M[i-1, j+2^{i-1}], \text{ ngược lại} \end{cases}$



CẤU TRÚC TRUY VẤN PHẦN TỬ NHỎ NHẤT TRÊN ĐOẠN CON (RMQ)

- Bài toán con nhỏ nhất M[0, *j*] = *j*, *j* = 0,…, *N*-1
- Công thức truy hồi
- $M[i, j] = \begin{cases} M[i-1, j] \text{ nếu } a[M[i-1, j]] < a[M[i-1, j+2^{i-1}]] \\ M[i-1, j+2^{i-1}], \text{ ngược lại} \end{cases}$

```
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)];
      }
    }
  }
}
```

CẤU TRÚC TRUY VẤN PHẦN TỬ NHỎ NHẤT TRÊN ĐOẠN CON (RMQ)

- Truy vấn **RMQ**(i,j)
  - k = [log(j-i+1)]
  - **RMQ**(i,j) = $M[k,i]$ nếu $a[M[k,i]] \leq a[M[k, j-2^k+1]]$
    $M[k, j-2^k+1]]$, ngược lại
- **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

CÂY PHÂN ĐOẠN (SEGMENT TREES)

- **Bài tập minh họa (P.02.03.02).** Cho dãy $a_1, a_2, \ldots, a_n$. Hãy thực hiện 1 dãy các thao tác sau đây trên dãy đã cho:
  - update i v: gán $a_i = v$
  - get-max i j: trả về giá trị lớn nhất trong dãy $a_i, a_{i+1}, \ldots, a_j$.

- **Bài tập minh họa (P.02.03.02).** Cho dãy $a_1, a_2, \ldots, a_n$. Hãy thực hiện 1 dãy các thao tác sau đây trên dãy đã cho:
    - update i v: gán $a_i = v$
    - get-max i j: trả về giá trị lớn nhất trong dãy $a_i, a_{i+1}, \ldots, a_j$.
- Thuật toán trực tiếp
    - Thao tác update i v: cập nhật $a_i = v$, độ phức tạp O(1)
    - Thao tác get-max i j: duyệt dãy $a_i, a_{i+1}, \ldots, a_j$ để tìm phần tử lớn nhất, độ phức tạp $O(j - i)$

CÂY PHÂN ĐOẠN (SEGMENT TREES)

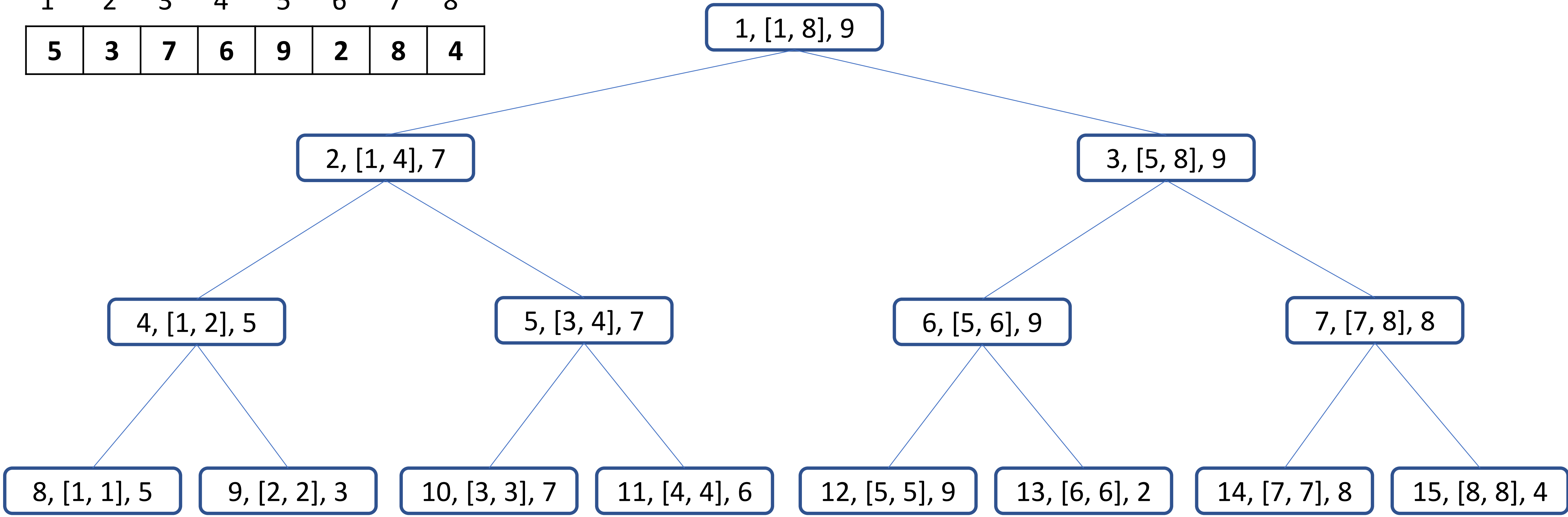
- Segment Trees: cấu trúc cây nhị phân đầy đủ
  - Mỗi nút quản lý 1 đoạn con trên cây
  - Nút gốc có *id* = 1 quản lý đoạn với chỉ số [1, *N*]
  - Mỗi nút có *id* = *v* quản lý đoạn với chỉ số [*i*, *j*] thì
    - Con trái có *id* = 2*v* quản lý đoạn với chỉ số [*i*, (*i*+*j*)/2]
    - Con phải có *id* = 2*v*+1 quản lý đoạn với chỉ số [(*i*+*j*)/2+1, *j*]
- Cấu trúc dữ liệu mỗi nút của cây

  id, [*L*, *R*], *maxVal*[*id*]

  - *id*: chỉ số của nút
  - *L* và *R*: chỉ số bắt đầu và chỉ số kết thúc của dãy con $a_L, a_{L+1}, \ldots, a_R$ mà nút quản lý
  - *maxVal*[*id*]: giá trị lớn nhất của dãy $a_L, a_{L+1}, \ldots, a_R$ mà nó quản lý

CÂY PHÂN ĐOẠN (SEGMENT TREES)

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



CÂY PHÂN ĐOẠN (SEGMENT TREES)

```
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
}
```

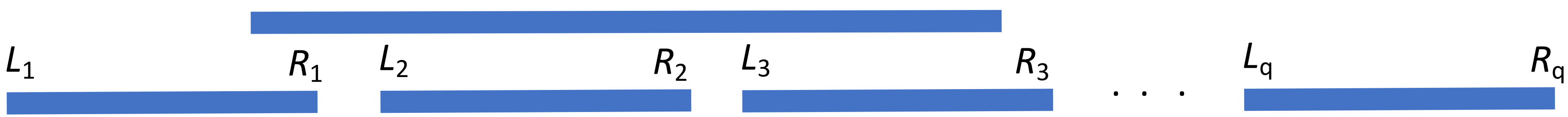
CÂY PHÂN ĐOẠN (SEGMENT TREES)

```
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
}
```

CÂY PHÂN ĐOẠN (SEGMENT TREES)

- Số lượng nút trên segment tree nhỏ hơn hoặc bằng 4n
  - Ký hiệu $k = \lceil \log N \rceil$
  - Số lượng nút trên cây nhiều nhất là $1 + 2^1 + 2^2 + \ldots + 2^k = 2^{k+1} - 1 < 4N$

- Phân tích độ phức tạp thao tác GetMax, ta sẽ duyệt qua nhiều nhất là 4 nút trên mỗi mức của cây (chứng minh bằng quy nạp)
  - Ở mức 1 (nút gốc): ta chỉ thăm 1 nút gốc
  - Giả sử ta đang ở một mức k hiện tại, ta thăm các nút V_k ($|V_k| \leq 4$)
    - Ta gọi 2 đoạn [a, b] và [c, d] là **over-lap** với nhau nếu đoạn này không phải là đoạn con (hoặc trùng khớp) của đoạn kia
    - Lưu ý: ở hàm GetMaxFromNode(*id*, *L*, *R*, *i*, *j*) tại nút (*id*, *L*, *R*), ta chỉ gọi đệ quy để đi thăm nút con nếu đoạn [*L*, *R*] và [*i*, *j*] là over-lap.
    - Giả sử các nút trong V_k (đi từ trái qua phải) là (id_1, L_1, R_1), (id_2, L_2, R_2), ..., (id_q, L_q, R_q). Rõ ràng, số đoạn trong [L_1, R_1], . . ., [L_q, R_q] over-lap với đoạn [*i*, *j*] phải nhỏ hơn hoặc bằng 2 được vì nếu ngược lại thì các nằm đoạn giữa trong dãy các đoạn over-lap với [*i*, *j*] này chắc chắn sẽ là đoạn con (hoặc trùng khớp) của đoạn [*i*, *j*] và vì thế từ các nút ứng với các đoạn ở giữa sẽ không gọi đệ quy để đi thăm nút con. Từ đó suy ra số nút con ở mức k+1 được thăm nhỏ hơn hoặc bằng 4



CÂY PHÂN ĐOẠN (SEGMENT TREES)

- Phân tích độ phức tạp thao tác GetMax, ta sẽ duyệt qua nhiều nhất là 4 nút trên mỗi mức của cây (chứng minh bằng quy nạp)
- Độ cao của cây là O(log*N*). Như vậy độ phức tạp của thao tác GetMax(*i*, *j*) là 4 x O(log*N*) hay là O(log*N*)

CÂY PHÂN ĐOẠN (SEGMENT TREES)

- Phân tích độ phức tạp thao tác Update(*i*, *v*)
  - Xuất phát từ nút gốc, tại mỗi mức *k*, ta chỉ thăm nhiều nhất là 1 nút con vì chỉ số *i* chỉ thuộc về nhiều nhất 1 đoạn con trong số 2 đoạn con được chia ra từ đoạn ở mức trước.
  - Do đó, độ phức tạp của thao tác Update(*i*, *v*) là độ cao của cây và là O(log*N*)

THANK YOU !