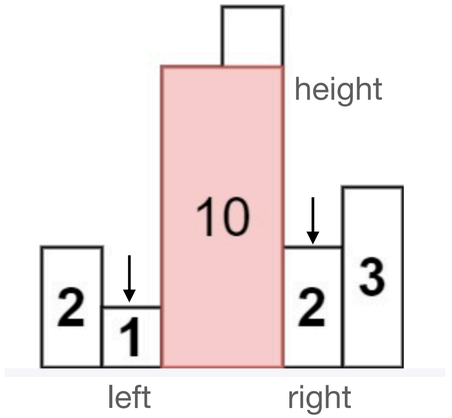
Largest Rectangle in Histogram

Tolerance

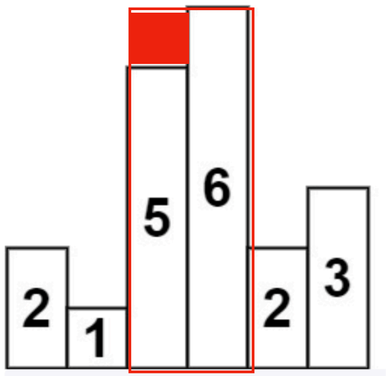
1. 알고리즘

가장 큰 직사각형의 후보들은 다음 세가지 요소로 구성됩니다.

- 왼쪽 끝 index: left
- 오른쪽 끝 index: right
- 직사각형 높이: height



<가장 작은 높이 선택>



<6인 높이를 택할 경우>

이때, 직사각형의 높이는 왼쪽 끝부터 오른쪽 끝 사이의 bar 중 가장 작은 높이가 됩니다.

(min(heights[i]) where left < i < right)

가장 작은 높이를 택하지 않으면, 가장 작은 높이의 bar를 넘어서는 영역을 후보 영역으로 포함하기 때문입니다.

넓이는 다음과 같이 구할 수 있습니다: (right - left - 1) * height

참고: a < x < b인 정수의 갯수: b - a - 1;

왼쪽 끝 구하기

```
vector<int> leftEnd(n, 0);
leftEnd[0] = -1;

for (int i = 1; i < n; i++)
{
    int j =  i - 1;
    while (j >= 0 && heights[j] >= heights[i])
    {
        j = leftEnd[j];
    }
    leftEnd[i] = j;
}
```

(1) 주어진 heights 벡터의 size를 n이라 할때, size n 을 가지는 leftEnd 벡터를 생성하고 초기화합니다.

- leftEnd[i]는 heights[i]를 가장 작은 높이로 가지는 후보 직사각형의 왼쪽 끝 index 값입니다.

(2) leftEnd[0] = -1로 초기화 합니다. 첫번째 후보의 왼쪽 끝 index를 나타내는 값입니다.

(3) leftEnd 벡터를 채워나갑니다.

- 현재 위치 i의 높이가 i이전의 j 위치의 높이보다 작거나 같다면, i의 왼쪽 끝은 j의 왼쪽 끝과 같거나 더 멀리에 위치할 것입니다. (즉, heights[j] >= heights[i]이면, leftEnd[j] >= leftEnd[i])

- 위의 정보를 이용하여, j에서의 높이가 현재 위치 i의 높이보다 같거나 클 경우 j의 왼쪽 끝으로 점프하여 탐색을 계속합니다.

오른쪽 끝 구하기

```
vector<int> rightEnd(n, 0);
rightEnd[n - 1] = n;

for (int i = n - 2; i >= 0; i--)
{
    int j = i + 1;
    while (j < n && heights[i] <= heights[j])
    {
        j = rightEnd[j];
    }
    rightEnd[i] = j;
}
```

(1) rightEnd 벡터도 leftEnd 벡터와 같은 방식으로 생성합니다.

- rightEnd[i]는 heights[i]를 가장 작은 높이로 가지는 후보 직사각형의 오른쪽 끝 index 값입니다.

(2) rightEnd[n - 1] = n으로 초기화 합니다. 오른쪽 끝에 위치한 후보 직사각형의 오른쪽 끝 값입니다.

(3) rightEnd 벡터를 채워나갑니다.

- 현재 위치 i의 높이가 i 이후의 j 위치의 높이보다 작거나 같다면, i의 오른쪽 끝은 j의 오른쪽 끝과 같거나 더 멀리에 위치할 것입니다. (즉, heights[j] >= heights[i]이면, rightEnd[j] <= rightEnd[i])

- 위의 정보를 이용하여, j에서의 높이가 현재 위치 i의 높이보다 같거나 클 경우 j의 오른쪽 끝으로 점프하여 탐색을 계속합니다.

넓이 구하기

```
int res = 0;

for (int i = 0; i < n; i++)
{
    res = max(res, heights[i] * (rightEnd[i] - leftEnd[i] - 1));
}
```

leftEnd, rightEnd 정보를 바탕으로 후보 중 최대 직사각형의 넓이를 구합니다.

2. 공간 복잡도

Input vector heights의 사이즈를 n이라 할때,

- size n의 leftEnd vector
- size n의 rightEnd vector
- int n, res

=> O(n)의 공간 복잡도.

3. 시간 복잡도

```
vector<int> leftEnd(n, 0);
leftEnd[0] = -1;

for (int i = 1; i < n; i++)
{
    int j =  i - 1;
    while (j >= 0 && heights[j] >= heights[i])
    {
        j = leftEnd[j];
    }
    leftEnd[i] = j;
}
```

배열에 얼마나 접근 하는지를 통해서 시간복잡도를 구할 수 있습니다. 왼쪽 끝 구하기로 예를 들겠습니다.

(1) 모든 위치에서 점프는 한번만 발생할 수 있습니다.

- 현재 위치가 i이고, i의 왼쪽 끝을 찾기 위해 i 이전의 j위치를 탐색하는 과정 중 점프가 발생했다면, 결과 값인 leftEnd[i]는 leftEnd[j]의 점프 정보를 포함하고 있습니다. 이후 부터는 leftEnd[j]를 방문하지 않으며, leftEnd[i]의 값을 점프 정보로 활용하게 됩니다. => O(n)의 점프 발생

(2) 점프 이외의 작업은 상수시간 작업이며 for loop는 n -1 번 수행됩니다.

- index i, j 계산, leftEnd[i] 입력

=> (1) + (2) = O(n) + O(n) = O(n)

오른쪽 끝 구하기도 위와 같은 방식이며, 넓이 계산 또한 O(n)의 시간 복잡도를 가집니다.