Data Structures and Algorithms

Searching and linear structures

October 27, 2023

Exercise 1.

find a way to calculate c^n in $O(lgn)$ time.

Solution 1.

```
def power (c, n):
    if n==1:
        return c
    if n % 2 == 1:
        return c * pow(c*c, n//2)
    else:
        return pow(c*c, n//2)
```

Exercise 2.

you are given a sorted array $A = [a_0, a_1, ..., a_i, ..., a_{n-1}, a_n]$. find an algorithm to find a_i with time $O(lg(i))$.

Solution 2.

```
def exponentialSearch(Array, x):
    bound = 1
    # find the range in which key `x` would reside
    while bound < len(Array) and Array[bound] < x:
        bound *= 2        # calculate the next power of 2

    # call binary search on A[bound/2 … min(bound, n-1)]
    return binarySearch(Array, bound // 2, min(bound, len(Array) - 1), x)
```

Exercise 3.

Show how to implement a stack using two queues. Analyze the running time of the stack operations.

Solution 3.

he idea is to keep newly entered element at the front of ‘q1’ so that pop operation dequeues from ‘q1’. ‘q2’ is used to put every new element in front of ‘q1’.

Follow the below steps to implement the push(s, x) operation:
1-Enqueue x to q2.
2-One by one dequeue everything from q1 and enqueue to q2.
3-Swap the queues of q1 and q2.

Follow the below steps to implement the pop(s) operation:
Dequeue an item from q1 and return it.

Exercise 4.

You are given an Array A of size N and Q questions about this array, each question contains 2 integers L and R which is a range and you have to find $a_L + a_{L+1}...a_R$.

find a solution in $O(N + Q)$

Solution 4.

$sum_i = a_0 + a_1 + ... + a_i$ $sum_{L,R} = sum_R - sum_{L-1}$)

Exercise 5.

same as previous question but now queries ask about the minimum element in the given range.

Exercise 6.

Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it can trap after raining.

Solution 6.

- Use stack to store the indices of the bars.
- Iterate the array:
  - While stack is not empty and $\text{height[current]} > \text{height[st.top()]}$
    - It means that the stack element can be popped. Pop the top element as top.
    - Find the distance between the current element and the element at top of stack, which is to be filled. $\text{distance} = \text{current} - \text{st.top()} - 1$
    - Find the bounded height $\text{bounded\_height} = \min(\text{height[current]}, \text{height[st.top()]}) - \text{height[top]}$
    - Add resulting trapped water to answer $\text{ans} \mathrel{+}= \text{distance} \times \text{bounded\_height}$
  - Push current index to top of the stack
  - Move current to the next position

time complexity $O(n)$