Deques Applications

Ulises Mdz, Ulises Tirado Zatarain

Algorist Weekly Talks

ulisesmdzmtz@gmail.com
ulises.tirado@cimat.mx

July 12, 2016

Overview

Finding Maximum

Description

Given an array **nums**, there is a *sliding window* of size **k** which is moving from the very left of the array to the very right. You can only see the **k** numbers in the window. Each time the sliding window moves right by one position, can you calculate the maximum element each time the window moves?. [a]

[a] Source: `https://leetcode.com/problems/sliding-window-maximum/`

Example/Simulation

For example, Given $\mathbf{nums} = [\mathbf{1}, \mathbf{3}, -\mathbf{1}, -\mathbf{3}, \mathbf{5}, \mathbf{3}, \mathbf{6}, \mathbf{7}]$, and $\mathbf{k} = \mathbf{3}$.

Window position								**Max**
[1	**3**	**-1]**	-3	5	3	6	7	**3**
1	**[3**	**-1**	**-3]**	5	3	6	7	**3**
1	3	**[-1**	**-3**	**5]**	3	6	7	**3**
1	3	-1	**[-3**	**5**	**3]**	6	7	**5**
1	3	-1	-3	**[5**	**3**	**6]**	7	**6**
1	3	-1	-3	5	**[3**	**6**	**7]**	**7**

Therefore, return the max sliding window as $[\mathbf{3}, \mathbf{3}, \mathbf{5}, \mathbf{5}, \mathbf{6}, \mathbf{7}]$.

Could you solve it in linear time?

C++ STL Deque Review

std::dueque

template < class T, class Alloc = allocator<T> > class deque;

Double ended queue

deque (usually pronounced like "deck") is an irregular acronym of double-ended queue. Double-ended queues are sequence containers with dynamic sizes that can be expanded or contracted on both ends (either its front or its back). Specific libraries may implement deques in different ways, generally as some form of dynamic array. But in any case, they allow for the individual elements to be accessed directly through random access iterators, with storage handled automatically by expanding and contracting the container as needed. Therefore, they provide a functionality similar to vectors, but with efficient insertion and deletion of elements also at the beginning of the sequence, and not only at its end.

Some things to notice

- What happens when we insert an element?
- What happens when we remove an element?
- What kind of data structure do we need?
- Do we need to store all elements from window?
- What we need to store data or index?
- How to know if some element belongs to some window?
- Useful elements

Solution

Using Deque

We create a **Deque**, $\mathbf{Q_i}$ of capacity **k**, that stores only *useful elements* of current window of **k** elements. **An element is useful** if it is in current window and is greater than all other elements on left side of it in current window. We process all array elements one by one and maintain **Qi** to contain useful elements of current window and these useful elements are maintained in sorted order. The element at front of the **Qi** is the largest and element at rear of **Qi** is the smallest of current window.

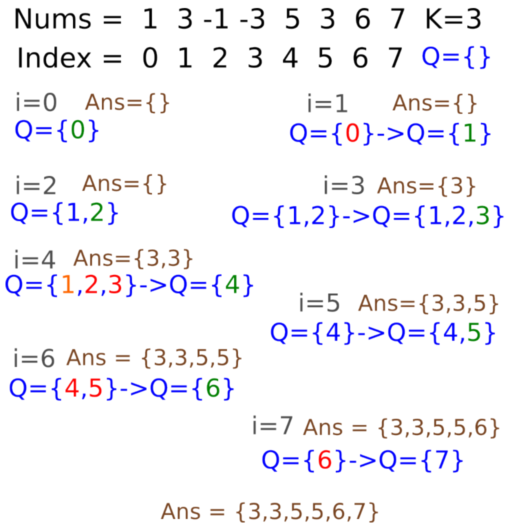
Solution Algorithm

Algorithm

- Process first **k** (or first window) elements of array.
  1. For very element, the previous smaller elements are useless so remove them from **Qi** (Remove form rear).
  2. Add new element at rear of queue.
- Process rest of the elements, i.e., from **k** to $\mathbf{n-1}$.
  1. The element at the front of the queue is **the largest element** of previous window, so print it.
  2. Remove the elements which are out of this window.
  3. Remove useless elements (all elements smaller than the currently being added).
  4. Add current element at the rear of **Qi**.
- Print the maximum element of last window

Full source code: `http://ideone.com/3a3rs9`

Visualization



Nums = 1 3 -1 -3 5 3 6 7 K=3
Index = 0 1 2 3 4 5 6 7 Q={}

i=0 Ans={}
Q={0}

i=1 Ans={}
Q={0}->Q={1}

i=2 Ans={}
Q={1,2}

i=3 Ans={3}
Q={1,2}->Q={1,2,3}

i=4 Ans={3,3}
Q={1,2,3}->Q={4}

i=5 Ans={3,3,5}
Q={4}->Q={4,5}

i=6 Ans = {3,3,5,5}
Q={4,5}->Q={6}

i=7 Ans = {3,3,5,5,6}
Q={6}->Q={7}

Ans = {3,3,5,5,6,7}

Figure : Algorithm to find maximum value on a slide window

Code Implementation (C++)

Example (C++ Implementation)

```
void printKMax(int arr[], int n, int k) {
   deque<int>  Qi;
   for(int i=0; i<k; ++i) {
      while ( (!Qi.empty()) && arr[i] >= arr[Qi.back()])
         Qi.pop_back();
      Qi.push_back(i);
   }
   for(int i=k; i<n; ++i) {
      cout << arr[Qi.front()] << "␣";
      while((!Qi.empty()) && Qi.front() <= (i-k))
         Qi.pop_front();
      while((!Qi.empty()) && arr[i]>=arr[Qi.back()])
         Qi.pop_back();
      Qi.push_back(i);
    }
    cout << arr[Qi.front()];
}
```

Q & A

Other solutions using DP

- http://stackoverflow.com/a/17249084
- http://www.zrzahid.com/sliding-window-minmax/

BFS 0/1

Ocean Currents (Difficult)

At each location, the current owes in some direction. The captain can choose to either go with the ow of the current, using no energy, or to move one square in any other direction, at the cost of one energy unit. The boat always moves in one of the following eight directions: north, south, east, west, north-east, Northwest, south-east, south-west. The boat cannot leave the boundary of the lake.
You are to help him devise a strategy to reach the destination with the minimum energy consumption.

References

- http://www.cplusplus.com/reference/queue/
- https://en.wikipedia.org/
- http://www.geeksforgeeks.org/maximum-of-all-subarrays-of-size-k/
- https://chococontest.wordpress.com/category/bfs-01/
- http://ideone.com/PvCked