INF280: Competitive programming

Advanced datastructure algorithms

Louis Jachiet

Sliding windows

Sliding window techniques to improve efficiency

Typical examples using a list $i_1, \ldots, i_N$ and an integer K

- find $max_j(i_j + \cdots + i_{j+K})$

Sliding window techniques to improve efficiency

Typical examples using a list $i_1, \ldots, i_N$ and an integer K

- find $max_j(i_j + \cdots + i_{j+K})$
- find $max_j\left(max\left(i_j, \ldots, i_{j+K}\right) - min\left(i_j, \ldots, i_{j+K}\right)\right)$

Sliding window techniques to improve efficiency

Typical examples using a list $i_1, \ldots, i_N$ and an integer K

- find $max_j(i_j + \cdots + i_{j+K})$
- find $max_j\left(max\left(i_j, \ldots, i_{j+K}\right) - min\left(i_j, \ldots, i_{j+K}\right)\right)$
- find $max_{j,\ell}\left(\ell - j \ \mid \ max(i_j, \ldots, i_\ell) - min(i_j, \ldots, i_\ell) < K\right)$

Sliding window techniques to improve efficiency

Typical examples using a list $i_1, \ldots, i_N$ and an integer K

- find $max_j(i_j + \cdots + i_{j+K})$
- find $max_j\,(max\,(i_j, \ldots, i_{j+K}) - min\,(i_j, \ldots, i_{j+K}))$
- find $max_{j,\ell}\,(\ell - j \;\mid\; max(i_j, \ldots, i_\ell) - min(i_j, \ldots, i_\ell) < K)$

Naive algorithm

Two (or three!) nested loops, recomputing from scratch for each position j.

Sliding window idea

Optimize away nested loops!

Example: fixed width (e.g. maintaining sum of K elements)

17	37	42	5	23	89	45	71	43	2	45	74	28	44	98

Sliding window techniques to improve efficiency

Sliding window idea

Optimize away nested loops!

Example: fixed width (e.g. maintaining sum of K elements)

17	37	42	5	23	89	45	71	43	2	45	74	28	44	98

Sliding window idea

Optimize away nested loops!

Example: fixed width (e.g. maintaining sum of K elements)

17	37	42	5	23	89	45	71	43	2	45	74	28	44	98

Sliding window idea

Optimize away nested loops!

Example: fixed width (e.g. maintaining sum of K elements)

17	37	42	5	23	89	45	71	43	2	45	74	28	44	98

Sliding window techniques to improve efficiency

Example: variable width (e.g. biggest total less than 100)

17	37	42	5	23	89	45	71	43	2	45	74	28	44	98

Sliding window techniques to improve efficiency

Example: variable width (e.g. biggest total less than 100)

17	37	42	5	23	89	45	71	43	2	45	74	28	44	98

Sliding window techniques to improve efficiency

Example: variable width (e.g. biggest total less than 100)

17	37	42	5	23	89	45	71	43	2	45	74	28	44	98

Sliding window techniques to improve efficiency

Example: variable width (e.g. biggest total less than 100)

17	37	42	5	23	89	45	71	43	2	45	74	28	44	98

Sliding window techniques to improve efficiency

Example: variable width (e.g. biggest total less than 100)

17	37	42	5	23	89	45	71	43	2	45	74	28	44	98

Sliding window techniques to improve efficiency

Example: variable width (e.g. biggest total less than 100)

17	37	42	5	23	89	45	71	43	2	45	74	28	44	98

Sliding window techniques to improve efficiency

Example: variable width (e.g. biggest total less than 100)

17	37	42	5	23	89	45	71	43	2	45	74	28	44	98

Sliding window techniques to improve efficiency

Example: variable width (e.g. biggest total less than 100)

17	37	42	5	23	89	45	71	43	2	45	74	28	44	98

Sliding window techniques to improve efficiency

Example: variable width (e.g. biggest total less than 100)

17	37	42	5	23	89	45	71	43	2	45	74	28	44	98

Sliding window techniques to improve efficiency

Example: variable width (e.g. biggest total less than 100)

17	37	42	5	23	89	45	71	43	2	45	74	28	44	98

Sliding window techniques to improve efficiency

Example: variable width (e.g. biggest total less than 100)

17	37	42	5	23	89	45	71	43	2	45	74	28	44	98

Sliding window techniques to improve efficiency

General idea

Maintain a double ended queue where:

- you add on the right to grow
- remove on the left to shrink
- maintain some computation over the window content

Works with monotone criteria for windows!

Sliding window techniques to improve efficiency

General idea

Maintain a double ended queue where:

- you add on the right to grow
- remove on the left to shrink
- maintain some computation over the window content

Works with monotone criteria for windows!

Advanced technique for sliding window

The deque trick to maintain min and max

Maintain the (ordered) list of elements that might become min/max.

Example: maintaining min of 5 elements)

17	37	42	5	23	89	45	71	43	2	35	74	28	44	98

Updates

- Add on the right: remove everything bigger
- Remove on the left: remove when min

The deque trick to maintain min and max

Maintain the (ordered) list of elements that might become min/max.

Example: maintaining min of 5 elements)

Candidate mins: 5, 23

17	37	42	5	23	89	45	71	43	2	35	74	28	44	98

Updates

- Add on the right: remove everything bigger
- Remove on the left: remove when min

The deque trick to maintain min and max

Maintain the (ordered) list of elements that might become min/max.

Example: maintaining min of 5 elements)

Candidate mins: 5, 23, 89

17	37	42	5	23	89	45	71	43	2	35	74	28	44	98

Updates

- Add on the right: remove everything bigger
- Remove on the left: remove when min

Advanced technique for sliding window

The deque trick to maintain min and max

Maintain the (ordered) list of elements that might become min/max.

Example: maintaining min of 5 elements)

Candidate mins: 5, 23, 45

17	37	42	5	23	89	45	71	43	2	35	74	28	44	98

Updates

- Add on the right: remove everything bigger
- Remove on the left: remove when min

Advanced technique for sliding window

The deque trick to maintain min and max

Maintain the (ordered) list of elements that might become min/max.

Example: maintaining min of 5 elements)

Candidate mins: 5, 23, 45, 71

17	37	42	5	23	89	45	71	43	2	35	74	28	44	98

Updates

- Add on the right: remove everything bigger
- Remove on the left: remove when min

Advanced technique for sliding window

The deque trick to maintain min and max

Maintain the (ordered) list of elements that might become min/max.

Example: maintaining min of 5 elements)

Candidate mins: 23, 43

17	37	42	5	23	89	45	71	43	2	35	74	28	44	98

Updates

- Add on the right: remove everything bigger
- Remove on the left: remove when min

Advanced technique for sliding window

The deque trick to maintain min and max

Maintain the (ordered) list of elements that might become min/max.

Example: maintaining min of 5 elements)

Candidate mins: 2

17	37	42	5	23	89	45	71	43	2	35	74	28	44	98

Updates

- Add on the right: remove everything bigger
- Remove on the left: remove when min

Advanced technique for sliding window

The deque trick to maintain min and max

Maintain the (ordered) list of elements that might become min/max.

Example: maintaining min of 5 elements)

Candidate mins: 2, 35

17	37	42	5	23	89	45	71	43	2	35	74	28	44	98

Updates

- Add on the right: remove everything bigger
- Remove on the left: remove when min

Advanced technique for sliding window

The deque trick to maintain min and max

Maintain the (ordered) list of elements that might become min/max.

Example: maintaining min of 5 elements)

Candidate mins: 2, 35,74

17	37	42	5	23	89	45	71	43	2	35	74	28	44	98

Updates

- Add on the right: remove everything bigger
- Remove on the left: remove when min

Advanced technique for sliding window

The deque trick to maintain min and max

Maintain the (ordered) list of elements that might become min/max.

Example: maintaining min of 5 elements)

17	37	42	5	23	89	45	71	43	2	35	74	28	44	98

Updates

- Add on the right: remove everything bigger
- Remove on the left: remove when min

Also works with a set... with a $O(\log(n))$ penalty.

Prefix sums

Structure to compute sums in $O(1)$

Input

A list of elements $v_1 ... v_n$ over a group

Query

Compute $q(i,j) = \sum_{i \leq \ell < j} v_\ell$

Structure to compute sums in $O(1)$

Input

A list of elements $v_1 ... v_n$ over a group

Query

Compute $q(i,j) = \sum_{i \leq \ell < j} v_\ell$

Solution

Precompute $T[i] = \sum_{\ell < i} v_i$, $q(i,j) = T[j] - T[i]$

Structure to compute sums in $O(1)$

Input

A list of elements $v_1 ... v_n$ over a group

Query

Compute $q(i, j) = \sum_{i \leq \ell < j} v_\ell$

Solution

Precompute $T[i] = \sum_{\ell < i} v_i$, $q(i, j) = T[j] - T[i]$

Works in d dimensions!