Algorithms

Lecture Topic: Online Algorithms (Part 2)

Anxiao (Andrew) Jiang

Roadmap of this lecture:

1. Understand "Online Algorithm" by solving the "Maintaining-a-Search-List Problem".

    1.1 Define the "Maintaining-a-Search-List Problem".

    1.2 Define the "Move-to-Front" algorithm.

    1.3 Competitive ratio of the "Move-to-Front" algorithm.

Maintaining a Search List

Task: maintain the order of elements in a linked list, so that the total cost of "searching for elements in the list" and "maintaining the list" is minimized.

Example of a linked list: $L = \langle 4, 1, 2, 3, 5 \rangle$

$$\Downarrow$$

$$4 \leftrightarrows 1 \leftrightarrows 2 \leftrightarrows 3 \leftrightarrows 5$$

Application of such a linked list: Hash table

Cost of "searching for element 2" = 3

Cost of "swapping two adjacent elements" = 1

Cost of "moving 2 to the front of the list" = 2

$$\langle 4, 1, 2, 3, 5 \rangle \Rightarrow \langle 4, 2, 1, 3, 5 \rangle \Rightarrow \langle 2, 4, 1, 3, 5 \rangle$$

Maintaining a Search List

Task: maintain the order of elements in a linked list, so that the total cost of "searching for elements in the list" and "maintaining the list" is minimized.

Linked list: $L = \langle x_1, x_2, \cdots, x_n \rangle$

$r_L(x)$: the position of element x in the list L. $\quad 1 \leq r_L(x) \leq n$

Cost of searching for element x in the list $L = r_L(x)$

Cost of swapping two adajcent elements = 1

Maintaining a Search List

Task: maintain the order of elements in a linked list, so that the total cost of "searching for elements in the list" and "maintaining the list" is minimized.

Why maintain the order of elements in the linked list (that is, to move elements around)?

Reason: we want to move elements that will be searched for frequently to the front of the list, to reduce the "search cost".

Issue: we do not know which elements will be searched for in the future!

Maintaining a Search List

Task: maintain the order of elements in a linked list, so that the total cost of "searching for elements in the list" and "maintaining the list" is minimized.

Why maintain the order of elements in the linked list (that is, to move elements around)?

Reason: we want to move elements that will be searched for frequently to the front of the list, to reduce the "search cost".

Issue: we do not know which elements will be searched for in the future!

Is it even possible to "maintain the list" in any useful way?

After all, no matter how we move elements, it is always possible that the next search is for the last element in the list!

Maintaining a Search List

Task: maintain the order of elements in a linked list, so that the total cost of "searching for elements in the list" and "maintaining the list" is minimized.

Why maintain the order of elements in the linked list (that is, to move elements around)?

Reason: we want to move elements that will be searched for frequently to the front of the list, to reduce the "search cost".

Issue: we do not know which elements will be searched for in the future!

Is it even possible to "maintain the list" in any useful way?

After all, no matter how we move elements, it is always possible that the next search is for the last element in the list!

Catch: our goal here is not to optimize the worst-case performance, but to minimize the "competitive ratio".

Maintaining a Search List

Task: maintain the order of elements in a linked list, so that the total cost of "searching for elements in the list" and "maintaining the list" is minimized.

Why maintain the order of elements in the linked list (that is, to move elements around)?

Reason: we want to move elements that will be searched for frequently to the front of the list, to reduce the "search cost".

Issue: we do not know which elements will be searched for in the future!

Is it even possible to "maintain the list" in any useful way?

After all, no matter how we move elements, it is always possible that the next search is for the last element in the list!

Catch: our goal here is not to optimize the worst-case performance, but to minimize the "competitive ratio".

If the optimal (minimum possible) cost is large, our algorithm's cost can also be large.
If the optimal cost is small, our algorithm's cost should also be small.
We just want their ratio always to be small:

$$\frac{\text{our cost}}{\text{optimal cost}}$$

Quiz questions:

1. What operations are allowed to maintain the search list?

2. What are the costs of those operations?

Roadmap of this lecture:

1. Understand "Online Algorithm" by solving the "Maintaining-a-Search-List Problem".

    1.1 Define the "Maintaining-a-Search-List Problem".

    1.2 Define the "Move-to-Front" algorithm.

    1.3 Competitive ratio of the "Move-to-Front" algorithm.

The online algorithm that we will explore: Move to Front

Idea: every time after an element is searched for, we move it to the front of the list.

The online algorithm that we will explore: Move to Front

Idea: every time after an element is searched for, we move it to the front of the list.

cost of "search for element x in the list L" $=$ search cost $+$ move cost $= r_L(x) + (r_L(x) - 1) = 2r_L(x) - 1$

The online algorithm that we will explore: Move to Front

Idea: every time after an element is searched for, we move it to the front of the list.

cost of "search for element x in the list L" $=$ search cost $+$ move cost $= r_L(x) + (r_L(x) - 1) = 2r_L(x) - 1$

Move-To-Front Algorithm

element searched	L	search cost	move cost	search +move cost	cumulative cost
	$\langle 1,2,3,4,5 \rangle$				

The online algorithm that we will explore: Move to Front

Idea: every time after an element is searched for, we move it to the front of the list.

cost of "search for element x in the list L" $=$ search cost $+$ move cost $= r_L(x) + (r_L(x) - 1) = 2r_L(x) - 1$

Move-To-Front Algorithm

element searched	L	search cost	move cost	search +move cost	cumulative cost
5	$\langle 1,2,3,4,5 \rangle$				

The online algorithm that we will explore: Move to Front

Idea: every time after an element is searched for, we move it to the front of the list.

cost of "search for element x in the list L" $=$ search cost $+$ move cost $= r_L(x) + (r_L(x) - 1) = 2r_L(x) - 1$

Move-To-Front Algorithm

element searched	L	search cost	move cost	search +move cost	cumulative cost
5	$\langle 1,2,3,4,5 \rangle$	5			

The online algorithm that we will explore: Move to Front

Idea: every time after an element is searched for, we move it to the front of the list.

cost of "search for element x in the list L" $=$ search cost $+$ move cost $= r_L(x) + (r_L(x) - 1) = 2r_L(x) - 1$

Move-To-Front Algorithm

element searched	L	search cost	move cost	search +move cost	cumulative cost
5	$\langle 1,2,3,4,5 \rangle$	5			
	$\langle 5,1,2,3,4 \rangle$				

The online algorithm that we will explore: Move to Front

Idea: every time after an element is searched for, we move it to the front of the list.

cost of "search for element x in the list L" $=$ search cost $+$ move cost $=$ $r_L(x) + (r_L(x) - 1) = 2r_L(x) - 1$

Move-To-Front Algorithm

element searched	L	search cost	move cost	search +move cost	cumulative cost
5	$\langle 1,2,3,4,5 \rangle$	5	4		
	$\langle 5,1,2,3,4 \rangle$				

The online algorithm that we will explore: Move to Front

Idea: every time after an element is searched for, we move it to the front of the list.

cost of "search for element x in the list L" $=$ search cost $+$ move cost $= r_L(x) + (r_L(x) - 1) = 2r_L(x) - 1$

Move-To-Front Algorithm

element searched	L	search cost	move cost	search +move cost	cumulative cost
5	$\langle 1,2,3,4,5 \rangle$	5	4	9	9
	$\langle 5,1,2,3,4 \rangle$				

The online algorithm that we will explore: Move to Front

Idea: every time after an element is searched for, we move it to the front of the list.

cost of "search for element x in the list L" $=$ search cost $+$ move cost $= r_L(x) + (r_L(x) - 1) = 2r_L(x) - 1$

Move-To-Front Algorithm

element searched	L	search cost	move cost	search +move cost	cumulative cost
5	$\langle 1,2,3,4,5 \rangle$	5	4	9	9
3	$\langle 5,1,2,3,4 \rangle$				

The online algorithm that we will explore: Move to Front

Idea: every time after an element is searched for, we move it to the front of the list.

cost of "search for element x in the list L" $=$ search cost $+$ move cost $= r_L(x) + (r_L(x) - 1) = 2r_L(x) - 1$

Move-To-Front Algorithm

element searched	L	search cost	move cost	search +move cost	cumulative cost
5	$\langle 1,2,3,4,5 \rangle$	5	4	9	9
3	$\langle 5,1,2,3,4 \rangle$	4			

The online algorithm that we will explore: Move to Front

Idea: every time after an element is searched for, we move it to the front of the list.

cost of "search for element x in the list L" $=$ search cost $+$ move cost $= r_L(x) + (r_L(x) - 1) = 2r_L(x) - 1$

Move-To-Front Algorithm

element searched	L	search cost	move cost	search +move cost	cumulative cost
5	$\langle 1,2,3,4,5 \rangle$	5	4	9	9
3	$\langle 5,1,2,3,4 \rangle$	4			
	$\langle 3,5,1,2,4 \rangle$				

The online algorithm that we will explore: Move to Front

Idea: every time after an element is searched for, we move it to the front of the list.

cost of "search for element x in the list L" $=$ search cost $+$ move cost $= r_L(x) + (r_L(x) - 1) = 2r_L(x) - 1$

Move-To-Front Algorithm

element searched	L	search cost	move cost	search +move cost	cumulative cost
5	$\langle 1,2,3,4,5 \rangle$	5	4	9	9
3	$\langle 5,1,2,3,4 \rangle$	4	3	7	16
	$\langle 3,5,1,2,4 \rangle$				

The online algorithm that we will explore: Move to Front

Idea: every time after an element is searched for, we move it to the front of the list.

cost of "search for element x in the list L" $=$ search cost $+$ move cost $= r_L(x) + (r_L(x) - 1) = 2r_L(x) - 1$

Move-To-Front Algorithm

element searched	L	search cost	move cost	search +move cost	cumulative cost
5	$\langle 1,2,3,4,5 \rangle$	5	4	9	9
3	$\langle 5,1,2,3,4 \rangle$	4	3	7	16
4	$\langle 3,5,1,2,4 \rangle$				

The online algorithm that we will explore: Move to Front

Idea: every time after an element is searched for, we move it to the front of the list.

cost of "search for element x in the list L" $=$ search cost $+$ move cost $= r_L(x) + (r_L(x) - 1) = 2r_L(x) - 1$

Move-To-Front Algorithm

element searched	L	search cost	move cost	search +move cost	cumulative cost
5	$\langle 1,2,3,4,5 \rangle$	5	4	9	9
3	$\langle 5,1,2,3,4 \rangle$	4	3	7	16
4	$\langle 3,5,1,2,4 \rangle$	5			

The online algorithm that we will explore: Move to Front

Idea: every time after an element is searched for, we move it to the front of the list.

cost of "search for element x in the list L" $=$ search cost $+$ move cost $= r_L(x) + (r_L(x) - 1) = 2r_L(x) - 1$

Move-To-Front Algorithm

element searched	L	search cost	move cost	search +move cost	cumulative cost
5	$\langle 1,2,3,4,5 \rangle$	5	4	9	9
3	$\langle 5,1,2,3,4 \rangle$	4	3	7	16
4	$\langle 3,5,1,2,4 \rangle$	5			
	$\langle 4,3,5,1,2 \rangle$				

The online algorithm that we will explore: Move to Front

Idea: every time after an element is searched for, we move it to the front of the list.

cost of "search for element x in the list L" $=$ search cost $+$ move cost $= r_L(x) + (r_L(x) - 1) = 2r_L(x) - 1$

Move-To-Front Algorithm

element searched	L	search cost	move cost	search +move cost	cumulative cost
5	$\langle 1,2,3,4,5 \rangle$	5	4	9	9
3	$\langle 5,1,2,3,4 \rangle$	4	3	7	16
4	$\langle 3,5,1,2,4 \rangle$	5	4	9	25
	$\langle 4,3,5,1,2 \rangle$				

The online algorithm that we will explore: Move to Front

Idea: every time after an element is searched for, we move it to the front of the list.

cost of "search for element x in the list L" $=$ search cost $+$ move cost $= r_L(x) + (r_L(x) - 1) = 2r_L(x) - 1$

Move-To-Front Algorithm

element searched	L	search cost	move cost	search +move cost	cumulative cost
5	$\langle 1,2,3,4,5 \rangle$	5	4	9	9
3	$\langle 5,1,2,3,4 \rangle$	4	3	7	16
4	$\langle 3,5,1,2,4 \rangle$	5	4	9	25
4	$\langle 4,3,5,1,2 \rangle$				

The online algorithm that we will explore: Move to Front

Idea: every time after an element is searched for, we move it to the front of the list.

cost of "search for element x in the list L" $=$ search cost $+$ move cost $= r_L(x) + (r_L(x) - 1) = 2r_L(x) - 1$

Move-To-Front Algorithm

element searched	L	search cost	move cost	search +move cost	cumulative cost
5	$\langle 1,2,3,4,5 \rangle$	5	4	9	9
3	$\langle 5,1,2,3,4 \rangle$	4	3	7	16
4	$\langle 3,5,1,2,4 \rangle$	5	4	9	25
4	$\langle 4,3,5,1,2 \rangle$	1	0	1	26

The online algorithm that we will explore: Move to Front

Idea: every time after an element is searched for, we move it to the front of the list.

cost of "search for element x in the list L" $=$ search cost $+$ move cost $= r_L(x) + (r_L(x) - 1) = 2r_L(x) - 1$

Move-To-Front Algorithm

element searched	L	search cost	move cost	search +move cost	cumulative cost
5	$\langle 1,2,3,4,5 \rangle$	5	4	9	9
3	$\langle 5,1,2,3,4 \rangle$	4	3	7	16
4	$\langle 3,5,1,2,4 \rangle$	5	4	9	25
4	$\langle 4,3,5,1,2 \rangle$	1	0	1	26

Foresee: an algorithm that knows the future inputs, and therefore can move elements optimally to minimize the total cost.

What will the Foresee algorithm do in this example?

The online algorithm that we will explore: Move to Front

Idea: every time after an element is searched for, we move it to the front of the list.

cost of "search for element x in the list L" $=$ search cost $+$ move cost $= r_L(x) + (r_L(x) - 1) = 2r_L(x) - 1$

	Move-To-Front Algorithm					Foresee Algorithm				
element searched	L	search cost	move cost	search +move cost	cumulative cost	L	search cost	move cost	search +move cost	cumulative cost
5	$\langle 1,2,3,4,5 \rangle$	5	4	9	9	$\langle 1,2,3,4,5 \rangle$				
3	$\langle 5,1,2,3,4 \rangle$	4	3	7	16					
4	$\langle 3,5,1,2,4 \rangle$	5	4	9	25					
4	$\langle 4,3,5,1,2 \rangle$	1	0	1	26					

The online algorithm that we will explore: Move to Front

Idea: every time after an element is searched for, we move it to the front of the list.

cost of "search for element x in the list L" $=$ search cost $+$ move cost $= r_L(x) + (r_L(x) - 1) = 2r_L(x) - 1$

	Move-To-Front Algorithm					Foresee Algorithm				
element searched	L	search cost	move cost	search +move cost	cumulative cost	L	search cost	move cost	search +move cost	cumulative cost
5	$\langle 1,2,3,4,5 \rangle$	5	4	9	9	$\langle 1,2,3,4,5 \rangle$	5			
3	$\langle 5,1,2,3,4 \rangle$	4	3	7	16					
4	$\langle 3,5,1,2,4 \rangle$	5	4	9	25					
4	$\langle 4,3,5,1,2 \rangle$	1	0	1	26					

The online algorithm that we will explore: Move to Front

Idea: every time after an element is searched for, we move it to the front of the list.

cost of "search for element x in the list L" $=$ search cost $+$ move cost $= r_L(x) + (r_L(x) - 1) = 2r_L(x) - 1$

	Move-To-Front Algorithm					Foresee Algorithm				
element searched	L	search cost	move cost	search +move cost	cumulative cost	L	search cost	move cost	search +move cost	cumulative cost
5	$\langle 1,2,3,4,5 \rangle$	5	4	9	9	$\langle 1,2,3,4,5 \rangle$	5			
3	$\langle 5,1,2,3,4 \rangle$	4	3	7	16	$\langle 1,2,3,4,5 \rangle$				
4	$\langle 3,5,1,2,4 \rangle$	5	4	9	25					
4	$\langle 4,3,5,1,2 \rangle$	1	0	1	26					

The online algorithm that we will explore: Move to Front

Idea: every time after an element is searched for, we move it to the front of the list.

cost of "search for element x in the list L" $=$ search cost $+$ move cost $= r_L(x) + (r_L(x) - 1) = 2r_L(x) - 1$

	Move-To-Front Algorithm					Foresee Algorithm				
element searched	L	search cost	move cost	search +move cost	cumulative cost	L	search cost	move cost	search +move cost	cumulative cost
5	$\langle 1,2,3,4,5 \rangle$	5	4	9	9	$\langle 1,2,3,4,5 \rangle$	5	0	5	5
3	$\langle 5,1,2,3,4 \rangle$	4	3	7	16	$\langle 1,2,3,4,5 \rangle$				
4	$\langle 3,5,1,2,4 \rangle$	5	4	9	25					
4	$\langle 4,3,5,1,2 \rangle$	1	0	1	26					

The online algorithm that we will explore: Move to Front

Idea: every time after an element is searched for, we move it to the front of the list.

cost of "search for element x in the list L" $=$ search cost $+$ move cost $=$ $r_L(x) + (r_L(x) - 1) = 2r_L(x) - 1$

	Move-To-Front Algorithm					Foresee Algorithm				
element searched	L	search cost	move cost	search +move cost	cumulative cost	L	search cost	move cost	search +move cost	cumulative cost
5	$\langle 1,2,3,4,5 \rangle$	5	4	9	9	$\langle 1,2,3,4,5 \rangle$	5	0	5	5
3	$\langle 5,1,2,3,4 \rangle$	4	3	7	16	$\langle 1,2,3,4,5 \rangle$	3			
4	$\langle 3,5,1,2,4 \rangle$	5	4	9	25					
4	$\langle 4,3,5,1,2 \rangle$	1	0	1	26					

The online algorithm that we will explore: Move to Front

Idea: every time after an element is searched for, we move it to the front of the list.

cost of "search for element x in the list L" $=$ search cost $+$ move cost $= r_L(x) + (r_L(x) - 1) = 2r_L(x) - 1$

Move-To-Front Algorithm | Foresee Algorithm

element searched	L	search cost	move cost	search +move cost	cumulative cost	L	search cost	move cost	search +move cost	cumulative cost
5	$\langle 1,2,3,4,5 \rangle$	5	4	9	9	$\langle 1,2,3,4,5 \rangle$	5	0	5	5
3	$\langle 5,1,2,3,4 \rangle$	4	3	7	16	$\langle 1,2,3,4,5 \rangle$	3			
4	$\langle 3,5,1,2,4 \rangle$	5	4	9	25	$\langle 4,1,2,3,5 \rangle$				
4	$\langle 4,3,5,1,2 \rangle$	1	0	1	26					

The online algorithm that we will explore: Move to Front

Idea: every time after an element is searched for, we move it to the front of the list.

cost of "search for element x in the list L" $=$ search cost $+$ move cost $= r_L(x) + (r_L(x) - 1) = 2r_L(x) - 1$

Move-To-Front Algorithm

element searched	L	search cost	move cost	search +move cost	cumulative cost
5	$\langle 1,2,3,4,5 \rangle$	5	4	9	9
3	$\langle 5,1,2,3,4 \rangle$	4	3	7	16
4	$\langle 3,5,1,2,4 \rangle$	5	4	9	25
4	$\langle 4,3,5,1,2 \rangle$	1	0	1	26

Foresee Algorithm

element searched	L	search cost	move cost	search +move cost	cumulative cost
5	$\langle 1,2,3,4,5 \rangle$	5	0	5	5
3	$\langle 1,2,3,4,5 \rangle$	3	3	6	11
4	$\langle 4,1,2,3,5 \rangle$				
4					

The online algorithm that we will explore: Move to Front

Idea: every time after an element is searched for, we move it to the front of the list.

cost of "search for element x in the list L" $=$ search cost $+$ move cost $= r_L(x) + (r_L(x) - 1) = 2r_L(x) - 1$

	Move-To-Front Algorithm					Foresee Algorithm				
element searched	L	search cost	move cost	search +move cost	cumulative cost	L	search cost	move cost	search +move cost	cumulative cost
5	$\langle 1,2,3,4,5\rangle$	5	4	9	9	$\langle 1,2,3,4,5\rangle$	5	0	5	5
3	$\langle 5,1,2,3,4\rangle$	4	3	7	16	$\langle 1,2,3,4,5\rangle$	3	3	6	11
4	$\langle 3,5,1,2,4\rangle$	5	4	9	25	$\langle 4,1,2,3,5\rangle$	1			
4	$\langle 4,3,5,1,2\rangle$	1	0	1	26					

The online algorithm that we will explore: Move to Front

Idea: every time after an element is searched for, we move it to the front of the list.

cost of "search for element x in the list L" $=$ search cost $+$ move cost $= r_L(x) + (r_L(x) - 1) = 2r_L(x) - 1$

	Move-To-Front Algorithm					Foresee Algorithm				
element searched	L	search cost	move cost	search +move cost	cumulative cost	L	search cost	move cost	search +move cost	cumulative cost
5	$\langle 1,2,3,4,5 \rangle$	5	4	9	9	$\langle 1,2,3,4,5 \rangle$	5	0	5	5
3	$\langle 5,1,2,3,4 \rangle$	4	3	7	16	$\langle 1,2,3,4,5 \rangle$	3	3	6	11
4	$\langle 3,5,1,2,4 \rangle$	5	4	9	25	$\langle 4,1,2,3,5 \rangle$	1			
4	$\langle 4,3,5,1,2 \rangle$	1	0	1	26	$\langle 4,1,2,3,5 \rangle$				

The online algorithm that we will explore: Move to Front

Idea: every time after an element is searched for, we move it to the front of the list.

cost of "search for element x in the list L" $=$ search cost $+$ move cost $= r_L(x) + (r_L(x) - 1) = 2r_L(x) - 1$

	Move-To-Front Algorithm					Foresee Algorithm				
element searched	L	search cost	move cost	search +move cost	cumulative cost	L	search cost	move cost	search +move cost	cumulative cost
5	$\langle 1,2,3,4,5 \rangle$	5	4	9	9	$\langle 1,2,3,4,5 \rangle$	5	0	5	5
3	$\langle 5,1,2,3,4 \rangle$	4	3	7	16	$\langle 1,2,3,4,5 \rangle$	3	3	6	11
4	$\langle 3,5,1,2,4 \rangle$	5	4	9	25	$\langle 4,1,2,3,5 \rangle$	1	0	1	12
4	$\langle 4,3,5,1,2 \rangle$	1	0	1	26	$\langle 4,1,2,3,5 \rangle$				

The online algorithm that we will explore: Move to Front

Idea: every time after an element is searched for, we move it to the front of the list.

cost of "search for element x in the list L" $=$ search cost $+$ move cost $= r_L(x) + (r_L(x) - 1) = 2r_L(x) - 1$

	Move-To-Front Algorithm					Foresee Algorithm				
element searched	L	search cost	move cost	search +move cost	cumulative cost	L	search cost	move cost	search +move cost	cumulative cost
5	$\langle 1,2,3,4,5 \rangle$	5	4	9	9	$\langle 1,2,3,4,5 \rangle$	5	0	5	5
3	$\langle 5,1,2,3,4 \rangle$	4	3	7	16	$\langle 1,2,3,4,5 \rangle$	3	3	6	11
4	$\langle 3,5,1,2,4 \rangle$	5	4	9	25	$\langle 4,1,2,3,5 \rangle$	1	0	1	12
4	$\langle 4,3,5,1,2 \rangle$	1	0	1	26	$\langle 4,1,2,3,5 \rangle$	1			

The online algorithm that we will explore: Move to Front

Idea: every time after an element is searched for, we move it to the front of the list.

cost of "search for element x in the list L" $=$ search cost $+$ move cost $= r_L(x) + (r_L(x) - 1) = 2r_L(x) - 1$

	Move-To-Front Algorithm					Foresee Algorithm				
element searched	L	search cost	move cost	search +move cost	cumulative cost	L	search cost	move cost	search +move cost	cumulative cost
5	$\langle 1,2,3,4,5 \rangle$	5	4	9	9	$\langle 1,2,3,4,5 \rangle$	5	0	5	5
3	$\langle 5,1,2,3,4 \rangle$	4	3	7	16	$\langle 1,2,3,4,5 \rangle$	3	3	6	11
4	$\langle 3,5,1,2,4 \rangle$	5	4	9	25	$\langle 4,1,2,3,5 \rangle$	1	0	1	12
4	$\langle 4,3,5,1,2 \rangle$	1	0	1	26	$\langle 4,1,2,3,5 \rangle$	1	0	1	13

We will prove: the Move-To-Front Algorithm has a competitive ratio of 4.

Quiz questions:

1. What is the "Move-to-Front" algorithm?

2. Do we know the optimal solution to the "Maintaining-a-Search-List Problem”?

Roadmap of this lecture:

1. Understand "Online Algorithm" by solving the "Maintaining-a-Search-List Problem".

    1.1 Define the "Maintaining-a-Search-List Problem".

    1.2 Define the "Move-to-Front" algorithm.

    1.3 Competitive ratio of the "Move-to-Front" algorithm.

We will prove: the Move-To-Front Algorithm has a competitive ratio of 4.

Inversion: a pair of elements, say a and b, in which a appears before b in one list, but b appears before a in another list.

We will prove: the Move-To-Front Algorithm has a competitive ratio of 4.

Inversion: a pair of elements, say a and b, in which a appears before b in one list, but b appears before a in another list.

Inversion count: For two lists L and L', let the inversion count $I(L, L')$ denote the number of inversions between the two lists, that is, the number of pairs of elements whose order differs in the two lists.

We will prove: the Move-To-Front Algorithm has a competitive ratio of 4.

Inversion: a pair of elements, say a and b, in which a appears before b in one list, but b appears before a in another list.

Inversion count: For two lists L and L', let the inversion count $I(L, L')$ denote the number of inversions between the two lists, that is, the number of pairs of elements whose order differs in the two lists.

Example: $L = \langle\ 5, 3, 1, 4, 2\ \rangle$ $\quad L' = \langle\ 3, 1, 2, 4, 5\ \rangle$

We will prove: the Move-To-Front Algorithm has a competitive ratio of 4.

Inversion: a pair of elements, say a and b, in which a appears before b in one list, but b appears before a in another list.

Inversion count: For two lists L and L', let the inversion count $I(L, L')$ denote the number of inversions between the two lists, that is, the number of pairs of elements whose order differs in the two lists.

Example: $L = \langle\, 5, 3, 1, 4, 2 \,\rangle$ $\qquad$ $L' = \langle\, 3, 1, 2, 4, 5 \,\rangle$

Out of the $\binom{5}{2} = 10$ pairs:

(1,2)	(2,4)
(1,3)	(2,5)
(1,4)	(3,4)
(1,5)	(3,5)
(2,3)	(4,5)

We will prove: the Move-To-Front Algorithm has a competitive ratio of 4.

Inversion: a pair of elements, say a and b, in which a appears before b in one list, but b appears before a in another list.

Inversion count: For two lists L and L', let the inversion count $I(L, L')$ denote the number of inversions between the two lists, that is, the number of pairs of elements whose order differs in the two lists.

Example: $L = \langle\, 5, 3, 1, 4, 2 \,\rangle$ $\qquad L' = \langle\, 3, 1, 2, 4, 5 \,\rangle$

Out of the $\binom{5}{2} = 10$ pairs:

(1,2)	(2,4)
(1,3)	(2,5)
(1,4)	(3,4)
(1,5)	(3,5)
(2,3)	(4,5)

there are 5 inversions.

$I(L, L') = 5$

We will prove: the Move-To-Front Algorithm has a competitive ratio of 4.

Inversion: a pair of elements, say a and b, in which a appears before b in one list, but b appears before a in another list.

Inversion count: For two lists L and L', let the inversion count $I(L, L')$ denote the number of inversions between the two lists, that is, the number of pairs of elements whose order differs in the two lists.

L_i^M : the list maintained by the Move-To-Front algorithm immediately after the i-th search.

We will prove: the Move-To-Front Algorithm has a competitive ratio of 4.

Inversion: a pair of elements, say a and b, in which a appears before b in one list, but b appears before a in another list.

Inversion count: For two lists L and L', let the inversion count $I(L, L')$ denote the number of inversions between the two lists, that is, the number of pairs of elements whose order differs in the two lists.

L_i^M : the list maintained by the Move-To-Front algorithm immediately after the i-th search.

L_i^F : the list maintained by the Foresee algorithm immediately after the i-th search.

We will prove: the Move-To-Front Algorithm has a competitive ratio of 4.

Inversion: a pair of elements, say a and b, in which a appears before b in one list, but b appears before a in another list.

Inversion count: For two lists L and L', let the inversion count $I(L, L')$ denote the number of inversions between the two lists, that is, the number of pairs of elements whose order differs in the two lists.

L_i^M : the list maintained by the Move-To-Front algorithm immediately after the i-th search.

L_i^F : the list maintained by the Foresee algorithm immediately after the i-th search.

c_i^M : the cost incurred by the Move-To-Front algorithm on its i-th call .

We will prove: the Move-To-Front Algorithm has a competitive ratio of 4.

Inversion: a pair of elements, say a and b, in which a appears before b in one list, but b appears before a in another list.

Inversion count: For two lists L and L', let the inversion count $I(L, L')$ denote the number of inversions between the two lists, that is, the number of pairs of elements whose order differs in the two lists.

L_i^M : the list maintained by the Move-To-Front algorithm immediately after the i-th search.

L_i^F : the list maintained by the Foresee algorithm immediately after the i-th search.

c_i^M : the cost incurred by the Move-To-Front algorithm on its i-th call .

c_i^F : the cost incurred by the Foresee algorithm on its i-th call .

We will prove: the Move-To-Front Algorithm has a competitive ratio of 4.

Inversion: a pair of elements, say a and b, in which a appears before b in one list, but b appears before a in another list.

Inversion count: For two lists L and L', let the inversion count $I(L, L')$ denote the number of inversions between the two lists, that is, the number of pairs of elements whose order differs in the two lists.

L_i^M : the list maintained by the Move-To-Front algorithm immediately after the i-th search.

L_i^F : the list maintained by the Foresee algorithm immediately after the i-th search.

c_i^M : the cost incurred by the Move-To-Front algorithm on its i-th call .

c_i^F : the cost incurred by the Foresee algorithm on its i-th call .

t_i : the number of swaps performed by the Foresee algorithm in its i-th call .

We will prove: the Move-To-Front Algorithm has a competitive ratio of 4.

Inversion: a pair of elements, say a and b, in which a appears before b in one list, but b appears before a in another list.

Inversion count: For two lists L and L', let the inversion count $I(L, L')$ denote the number of inversions between the two lists, that is, the number of pairs of elements whose order differs in the two lists.

L_i^M : the list maintained by the Move-To-Front algorithm immediately after the i-th search.

L_i^F : the list maintained by the Foresee algorithm immediately after the i-th search.

c_i^M : the cost incurred by the Move-To-Front algorithm on its i-th call .

c_i^F : the cost incurred by the Foresee algorithm on its i-th call .

t_i : the number of swaps performed by the Foresee algorithm in its i-th call .

If the i-th operation is a search for element x, then $c_i^M = 2r_{L_{i-1}^M}(x) - 1$ $\quad$ $c_i^F = r_{L_{i-1}^F}(x) + t_i$

In order to compare the costs more carefully, let's break down the elements into subsets, depending on their positions in the two lists before the i-th search, relative to the element x being search for in the i-th search. We define three sets:

$$\text{BB} = \{\text{elements before } x \text{ in both } L_{i-1}^M \text{ and } L_{i-1}^F\}$$

$$\text{BA} = \{\text{elements before } x \text{ in } L_{i-1}^M \text{ but after } x \text{ in } L_{i-1}^F\}$$

$$\text{AB} = \{\text{elements after } x \text{ in } L_{i-1}^M \text{ but before } x \text{ in } L_{i-1}^F\}$$

In order to compare the costs more carefully, let's break down the elements into subsets, depending on their positions in the two lists before the i-th search, relative to the element x being search for in the i-th search. We define three sets:

BB $=$ {elements before x in both L_{i-1}^M and L_{i-1}^F}

BA $=$ {elements before x in L_{i-1}^M but after x in L_{i-1}^F}

AB $=$ {elements after x in L_{i-1}^M but before x in L_{i-1}^F}

$$r_{L_{i-1}^M}(x) = |BB| + |BA| + 1$$

$$r_{L_{i-1}^F}(x) = |BB| + |AB| + 1$$

In order to compare the costs more carefully, let's break down the elements into subsets, depending on their positions in the two lists before the i-th search, relative to the element x being search for in the i-th search. We define three sets:

$$\text{BB} = \{\text{elements before } x \text{ in both } L^M_{i-1} \text{ and } L^F_{i-1}\}$$

$$\text{BA} = \{\text{elements before } x \text{ in } L^M_{i-1} \text{ but after } x \text{ in } L^F_{i-1}\}$$

$$\text{AB} = \{\text{elements after } x \text{ in } L^M_{i-1} \text{ but before } x \text{ in } L^F_{i-1}\}$$

$$r_{L^M_{i-1}}(x) = |BB| + |BA| + 1$$

$$r_{L^F_{i-1}}(x) = |BB| + |AB| + 1$$

Given two lists L and L', if we swap two adjacent elements in one list, the inversion count $I(L, L')$ either increases or decreases by 1.

In order to compare the costs more carefully, let's break down the elements into subsets, depending on their positions in the two lists before the i-th search, relative to the element x being search for in the i-th search. We define three sets:

$$\text{BB} = \{\text{elements before } x \text{ in both } L^M_{i-1} \text{ and } L^F_{i-1}\}$$

$$\text{BA} = \{\text{elements before } x \text{ in } L^M_{i-1} \text{ but after } x \text{ in } L^F_{i-1}\}$$

$$\text{AB} = \{\text{elements after } x \text{ in } L^M_{i-1} \text{ but before } x \text{ in } L^F_{i-1}\}$$

$$r_{L^M_{i-1}}(x) = |BB| + |BA| + 1$$

$$r_{L^F_{i-1}}(x) = |BB| + |AB| + 1$$

Given two lists L and L', if we swap two adjacent elements in one list, the inversion count $I(L, L')$ either increases or decreases by 1.

$$I(L^M_i, L^F_{i-1}) - I(L^M_{i-1}, L^F_{i-1}) = |BB| - |BA|$$

Theorem: Algorithm Move-To-Front has a competitive ratio of 4.

Theorem: Algorithm Move-To-Front has a competitive ratio of 4.

Proof: We use amortized analysis based on potential function.

Potential function: $\Phi_i = 2I(L_i^M, L_i^F)$

Note: $\Phi_0 = 0, \quad \Phi_i \geq 0$

Theorem: Algorithm Move-To-Front has a competitive ratio of 4.

Proof: We use amortized analysis based on potential function.

Potential function: $\Phi_i = 2I(L_i^M, L_i^F)$

Note: $\Phi_0 = 0, \quad \Phi_i \geq 0$

The amortized cost $\hat{c}_i^M$ of the i-th Move-To-Front operation is $\hat{c}_i^M = c_i^M + \Phi_i - \Phi_{i-1}$

Theorem: Algorithm Move-To-Front has a competitive ratio of 4.

Proof: We use amortized analysis based on potential function.

Potential function: $\Phi_i = 2I(L_i^M, L_i^F)$

Note: $\Phi_0 = 0, \ \Phi_i \geq 0$

The amortized cost $\hat{c}_i^M$ of the i-th Move-To-Front operation is $\hat{c}_i^M = c_i^M + \Phi_i - \Phi_{i-1}$

Each swap performed by the Foresee algorithm either increases or decreases the potential by 2.

Theorem: Algorithm Move-To-Front has a competitive ratio of 4.

Proof: We use amortized analysis based on potential function.

Potential function: $\Phi_i = 2I(L_i^M, L_i^F)$

Note: $\Phi_0 = 0, \ \ \Phi_i \geq 0$

The amortized cost $\hat{c}_i^M$ of the i-th Move-To-Front operation is $\hat{c}_i^M = c_i^M + \Phi_i - \Phi_{i-1}$

Each swap performed by the Foresee algorithm either increases or decreases the potential by 2.

$$\hat{c}_i^M = c_i^M + \Phi_i - \Phi_{i-1}$$

Theorem: Algorithm Move-To-Front has a competitive ratio of 4.

Proof: We use amortized analysis based on potential function.

Potential function: $\Phi_i = 2I(L_i^M, L_i^F)$ Note: $\Phi_0 = 0, \ \Phi_i \geq 0$

The amortized cost $\hat{c}_i^M$ of the i-th Move-To-Front operation is $\hat{c}_i^M = c_i^M + \Phi_i - \Phi_{i-1}$

Each swap performed by the Foresee algorithm either increases or decreases the potential by 2.

$$\hat{c}_i^M = c_i^M + \Phi_i - \Phi_{i-1} \leq 2\, r_{L_{i-1}^M}(x) - 1 + 2\,(|BB| - |BA| + t_i)$$

Theorem: Algorithm Move-To-Front has a competitive ratio of 4.

Proof: We use amortized analysis based on potential function.

Potential function: $\Phi_i = 2I(L_i^M, L_i^F)$ Note: $\Phi_0 = 0, \ \Phi_i \geq 0$

The amortized cost $\hat{c}_i^M$ of the i-th Move-To-Front operation is $\hat{c}_i^M = c_i^M + \Phi_i - \Phi_{i-1}$

Each swap performed by the Foresee algorithm either increases or decreases the potential by 2.

$$\hat{c}_i^M = c_i^M + \Phi_i - \Phi_{i-1} \leq 2\, r_{L_{i-1}^M}(x) - 1 + 2\,(|BB| - |BA| + t_i)$$

$$= 2\, r_{L_{i-1}^M}(x) - 1 + 2\,(|BB| - (r_{L_{i-1}^M}(x) - 1 - |BB|) + t_i)$$

because: $r_{L_{i-1}^M}(x) = |BB| + |BA| + 1$

Theorem: Algorithm Move-To-Front has a competitive ratio of 4.

Proof: We use amortized analysis based on potential function.

Potential function: $\Phi_i = 2I(L_i^M, L_i^F)$ Note: $\Phi_0 = 0, \ \Phi_i \geq 0$

The amortized cost $\hat{c}_i^M$ of the i-th Move-To-Front operation is $\hat{c}_i^M = c_i^M + \Phi_i - \Phi_{i-1}$

Each swap performed by the Foresee algorithm either increases or decreases the potential by 2.

$$\hat{c}_i^M = c_i^M + \Phi_i - \Phi_{i-1} \leq 2\, r_{L_{i-1}^M}(x) - 1 + 2\,(|BB| - |BA| + t_i)$$

$$= 2\, r_{L_{i-1}^M}(x) - 1 + 2\,(|BB| - (r_{L_{i-1}^M}(x) - 1 - |BB|) + t_i)$$

$$= 4|BB| + 1 + 2t_i$$

Theorem: Algorithm Move-To-Front has a competitive ratio of 4.

Proof: We use amortized analysis based on potential function.

Potential function: $\Phi_i = 2I(L_i^M, L_i^F)$ Note: $\Phi_0 = 0, \ \Phi_i \geq 0$

The amortized cost $\hat{c}_i^M$ of the i-th Move-To-Front operation is $\hat{c}_i^M = c_i^M + \Phi_i - \Phi_{i-1}$

Each swap performed by the Foresee algorithm either increases or decreases the potential by 2.

$$\hat{c}_i^M = c_i^M + \Phi_i - \Phi_{i-1} \leq 2\, r_{L_{i-1}^M}(x) - 1 + 2\,(|BB| - |BA| + t_i)$$

$$= 2\, r_{L_{i-1}^M}(x) - 1 + 2\,(|BB| - (r_{L_{i-1}^M}(x) - 1 - |BB|) + t_i)$$

$$= 4|BB| + 1 + 2t_i \leq 4|BB| + 4|AB| + 4 + 4t_i$$

Theorem: Algorithm Move-To-Front has a competitive ratio of 4.

Proof: We use amortized analysis based on potential function.

Potential function: $\Phi_i = 2I(L_i^M, L_i^F)$ Note: $\Phi_0 = 0, \ \Phi_i \geq 0$

The amortized cost $\hat{c}_i^M$ of the i-th Move-To-Front operation is $\hat{c}_i^M = c_i^M + \Phi_i - \Phi_{i-1}$

Each swap performed by the Foresee algorithm either increases or decreases the potential by 2.

$$\begin{aligned}
\hat{c}_i^M = c_i^M + \Phi_i - \Phi_{i-1} &\leq 2\, r_{L_{i-1}^M}(x) - 1 + 2\,(|BB| - |BA| + t_i) \\
&= 2\, r_{L_{i-1}^M}(x) - 1 + 2\,(|BB| - (r_{L_{i-1}^M}(x) - 1 - |BB|) + t_i) \\
&= 4|BB| + 1 + 2t_i \leq 4|BB| + 4|AB| + 4 + 4t_i \quad = 4(|BB| + |AB| + 1 + t_i)
\end{aligned}$$

Theorem: Algorithm Move-To-Front has a competitive ratio of 4.

Proof: We use amortized analysis based on potential function.

Potential function: $\Phi_i = 2I(L_i^M, L_i^F)$ Note: $\Phi_0 = 0, \ \Phi_i \geq 0$

The amortized cost $\hat{c}_i^M$ of the i-th Move-To-Front operation is $\hat{c}_i^M = c_i^M + \Phi_i - \Phi_{i-1}$

Each swap performed by the Foresee algorithm either increases or decreases the potential by 2.

$$
\begin{aligned}
\hat{c}_i^M &= c_i^M + \Phi_i - \Phi_{i-1} \leq 2\, r_{L_{i-1}^M}(x) - 1 + 2\,(|BB| - |BA| + t_i) \\
&= 2\, r_{L_{i-1}^M}(x) - 1 + 2\,(|BB| - (r_{L_{i-1}^M}(x) - 1 - |BB|) + t_i) \\
&= 4|BB| + 1 + 2t_i \leq 4|BB| + 4|AB| + 4 + 4t_i \quad = 4(|BB| + |AB| + 1 + t_i) \\
&= 4\,(r_{L_{i-1}^F}(x) + t_i)
\end{aligned}
$$

because: $r_{L_{i-1}^F}(x) = |BB| + |AB| + 1$

Theorem: Algorithm Move-To-Front has a competitive ratio of 4.

Proof: We use amortized analysis based on potential function.

Potential function: $\Phi_i = 2I(L_i^M, L_i^F)$ Note: $\Phi_0 = 0, \ \Phi_i \geq 0$

The amortized cost $\hat{c}_i^M$ of the i-th Move-To-Front operation is $\hat{c}_i^M = c_i^M + \Phi_i - \Phi_{i-1}$

Each swap performed by the Foresee algorithm either increases or decreases the potential by 2.

$$\hat{c}_i^M = c_i^M + \Phi_i - \Phi_{i-1} \leq 2\, r_{L_{i-1}^M}(x) - 1 + 2\,(|BB| - |BA| + t_i)$$

$$= 2\, r_{L_{i-1}^M}(x) - 1 + 2\,(|BB| - (r_{L_{i-1}^M}(x) - 1 - |BB|) + t_i)$$

$$= 4|BB| + 1 + 2t_i \leq 4|BB| + 4|AB| + 4 + 4t_i \quad = 4(|BB| + |AB| + 1 + t_i)$$

$$= 4\,(r_{L_{i-1}^F}(x) + t_i) \quad = 4\, c_i^F$$

Theorem: Algorithm Move-To-Front has a competitive ratio of 4.

Proof: We use amortized analysis based on potential function.

Potential function: $\Phi_i = 2I(L_i^M, L_i^F)$ Note: $\Phi_0 = 0, \ \Phi_i \geq 0$

The amortized cost $\hat{c}_i^M$ of the i-th Move-To-Front operation is $\hat{c}_i^M = c_i^M + \Phi_i - \Phi_{i-1}$

Each swap performed by the Foresee algorithm either increases or decreases the potential by 2.

$$\begin{aligned}
\hat{c}_i^M &= c_i^M + \Phi_i - \Phi_{i-1} \leq 2\, r_{L_{i-1}^M}(x) - 1 + 2\,(|BB| - |BA| + t_i) \\
&= 2\, r_{L_{i-1}^M}(x) - 1 + 2\,(|BB| - (r_{L_{i-1}^M}(x) - 1 - |BB|) + t_i) \\
&= 4|BB| + 1 + 2t_i \leq 4|BB| + 4|AB| + 4 + 4t_i = 4(|BB| + |AB| + 1 + t_i) \\
&= 4\,(r_{L_{i-1}^F}(x) + t_i) = 4\, c_i^F
\end{aligned}$$

$$\sum_{i=1}^{m} c_i^M \leq \sum_{i=1}^{m} c_i^M + \Phi_m - \Phi_0$$

Theorem: Algorithm Move-To-Front has a competitive ratio of 4.

Proof: We use amortized analysis based on potential function.

Potential function: $\Phi_i = 2I(L_i^M, L_i^F)$ Note: $\Phi_0 = 0, \quad \Phi_i \geq 0$

The amortized cost $\hat{c}_i^M$ of the i-th Move-To-Front operation is $\hat{c}_i^M = c_i^M + \Phi_i - \Phi_{i-1}$

Each swap performed by the Foresee algorithm either increases or decreases the potential by 2.

$$\hat{c}_i^M = c_i^M + \Phi_i - \Phi_{i-1} \leq 2\, r_{L_{i-1}^M}(x) - 1 + 2\,(|BB| - |BA| + t_i)$$

$$= 2\, r_{L_{i-1}^M}(x) - 1 + 2\,(|BB| - (r_{L_{i-1}^M}(x) - 1 - |BB|) + t_i)$$

$$= 4|BB| + 1 + 2t_i \leq 4|BB| + 4|AB| + 4 + 4t_i = 4(|BB| + |AB| + 1 + t_i)$$

$$= 4\,(r_{L_{i-1}^F}(x) + t_i) = 4\, c_i^F$$

$$\sum_{i=1}^{m} c_i^M \leq \sum_{i=1}^{m} c_i^M + \Phi_m - \Phi_0 = \sum_{i=1}^{m} (c_i^M + \Phi_i - \Phi_{i-1}) = \sum_{i=1}^{m} \hat{c}_i^M$$

Theorem: Algorithm Move-To-Front has a competitive ratio of 4.

Proof: We use amortized analysis based on potential function.

Potential function: $\Phi_i = 2I(L_i^M, L_i^F)$ Note: $\Phi_0 = 0, \ \Phi_i \geq 0$

The amortized cost $\hat{c}_i^M$ of the i-th Move-To-Front operation is $\hat{c}_i^M = c_i^M + \Phi_i - \Phi_{i-1}$

Each swap performed by the Foresee algorithm either increases or decreases the potential by 2.

$$\hat{c}_i^M = c_i^M + \Phi_i - \Phi_{i-1} \leq 2\, r_{L_{i-1}^M}(x) - 1 + 2\,(|BB| - |BA| + t_i)$$

$$= 2\, r_{L_{i-1}^M}(x) - 1 + 2\,(|BB| - (r_{L_{i-1}^M}(x) - 1 - |BB|) + t_i)$$

$$= 4|BB| + 1 + 2t_i \leq 4|BB| + 4|AB| + 4 + 4t_i = 4(|BB| + |AB| + 1 + t_i)$$

$$= 4\,(r_{L_{i-1}^F}(x) + t_i) = 4\, c_i^F$$

$$\sum_{i=1}^{m} c_i^M \leq \sum_{i=1}^{m} c_i^M + \Phi_m - \Phi_0 = \sum_{i=1}^{m} (c_i^M + \Phi_i - \Phi_{i-1}) = \sum_{i=1}^{m} \hat{c}_i^M \leq \sum_{i=1}^{m} 4\, c_i^F = 4 \sum_{i=1}^{m} c_i^F$$

Quiz question:

1. How did we find the competitive ratio of the "Move-to-Front" algorithm without knowing the optimal solution?