Online Algorithms

Md. Saidur Rahman,
Department of Computer Science and Engineering,
Bangladesh University of Engineering and Technology,
Dhaka, Bangladesh.

Online Algorithms

Online Algorithms

In *online computation*, an algorithm must produce a sequence of decisions that will have an impact on the final quality of its overall performance. Each of these decisions must be made based on past events without secure information about the future. Such an algorithm is called an *online algorithm*.

Online Algorithms

Online Algorithms

In *online computation*, an algorithm must produce a sequence of decisions that will have an impact on the final quality of its overall performance. Each of these decisions must be made based on past events without secure information about the future. Such an algorithm is called an *online algorithm*.

Online Algorithms

Online Algorithms

In *online computation*, an algorithm must produce a sequence of decisions that will have an impact on the final quality of its overall performance. Each of these decisions must be made based on past events without secure information about the future. Such an algorithm is called an *online algorithm*.

Require immediate decisions to be made in real time.
Examples:

- decision making in financial market

Online Algorithms

Online Algorithms

In *online computation*, an algorithm must produce a sequence of decisions that will have an impact on the final quality of its overall performance. Each of these decisions must be made based on past events without secure information about the future. Such an algorithm is called an *online algorithm*.

Require immediate decisions to be made in real time.
Examples:

- decision making in financial market
- Paging in virtual management system

Online Algorithms

Online Algorithms

In *online computation*, an algorithm must produce a sequence of decisions that will have an impact on the final quality of its overall performance. Each of these decisions must be made based on past events without secure information about the future. Such an algorithm is called an *online algorithm*.

Require immediate decisions to be made in real time.
Examples:

- decision making in financial market
- Paging in virtual management system
- Routing in communication networks.

Online Algorithms

Online Algorithms

Online Algorithms are algorithms that need to make decisions without full knowledge of the input . They have full knowledge of the past but no (or partial) knowledge of the future.

Online Algorithms

Online Algorithms

Online Algorithms are algorithms that need to make decisions without full knowledge of the input . They have full knowledge of the past but no (or partial) knowledge of the future.

Online Algorithms

Online Algorithms

Online Algorithms are algorithms that need to make decisions without full knowledge of the input . They have full knowledge of the past but no (or partial) knowledge of the future.

Comptitive Analysis

For this type of problem we will attempt to design algorithms that are competitive with the optimum offline algorithm, the algorithm that has perfect knowledge of the future.

Online Algorithms

Comptitive Analysis

For this type of problem we will attempt to design algorithms that are competitive with the optimum offline algorithm, the algorithm that has perfect knowledge of the future.

Online Algorithms

Comptitive Analysis

For this type of problem we will attempt to design algorithms that are competitive with the optimum offline algorithm, the algorithm that has perfect knowledge of the future.

Comptitive Ratio

$$CR(A) = \max_i \frac{A(i)}{Opt(i)}$$

Where
$A(i)$ = cost of algorithm A for input i.
$Opt(i)$ = the cost of the optimal offline algorithm for input i.

Online Algorithms

Ski Rental Problem

This is a rent/buy problem.
You are going skiing for an unknown number of days. Assume that renting skis costs 500 Taka per day and buying skis costs 1500 Taka. Every day you have to decide whether to continue renting skis for one more day or buy a pair of skis. If you know in advance how many days you will go skiing, you can decide your minimum cost. The question is what to do when you do not know in advance how many days you will ski.

Online Algorithms

Ski Rental Problem

This is a rent/buy problem.
You are going skiing for an unknown number of days. Assume that renting skis costs 500 Taka per day and buying skis costs 1500 Taka. Every day you have to decide whether to continue renting skis for one more day or buy a pair of skis. If you know in advance how many days you will go skiing, you can decide your minimum cost. The question is what to do when you do not know in advance how many days you will ski.

Online Algorithms

Ski Rental Problem

This is a rent/buy problem.
You are going skiing for an unknown number of days. Assume that renting skis costs 500 Taka per day and buying skis costs 1500 Taka. Every day you have to decide whether to continue renting skis for one more day or buy a pair of skis. If you know in advance how many days you will go skiing, you can decide your minimum cost. The question is what to do when you do not know in advance how many days you will ski.

What is the best algorithm?

Online Algorithms

Problem Formulation

There is a number of days d (unknown to you) that you will ski. We are looking for an algorithm that minimizes the ratio between what you pay using the algorithm (that does not know d) and what you would pay optimally if you knew d in advance. The problem is generally analyzed in the worst case, where the algorithm is fixed and then we look at the worst-case performance of the algorithm over all possible d.

What is the best algorithm?

Online Algorithms

Ski Rental

Input: $1, 1, 1, 1, \cdots$ (we do not know how many 1's)
Action: pay for single time or go for free drive

Online Algorithms

Ski Rental

Input: $1, 1, 1, 1, \cdots$ (we do not know how many 1's)
Action: pay for single time or go for free drive

Online Algorithms

Ski Rental

Input: $1, 1, 1, 1, \cdots$ (we do not know how many 1's)
Action: pay for single time or go for free drive

Algorithms

Algorithm 1: Always pay for one time.
Algorithm 2: free drive from the begining
Algorithm 3: pay per time until two 1's and when the third 1 comes then free drive.

Online Algorithms: Preformance Measure

Ski Rental: Optimal

$$Opt(1) = 500$$

$$Opt(11) = 1000$$

$$Opt(111) = 1500$$

$$Opt(111\cdots) = 1500$$

Online Algorithms: Preformance Measure

Ski Rental: Optimal

$$Opt(1) = 500$$

$$Opt(11) = 1000$$

$$Opt(111) = 1500$$

$$Opt(111\cdots) = 1500$$

Ski Rental: Algorithm 1

$A_1(1) = 500, A_1(11) = 1000, A_1(111) = 1500, A_1(1111) = 2000$

$$\frac{A_1(1)}{OPT(1)} = \frac{500}{500}, \frac{A_1(11)}{OPT(11)} = \frac{1000}{1000}, \frac{A_1(111)}{OPT(111)} = \frac{1500}{1500}, \frac{A_1(1111)}{OPT(1111)} = \frac{2000}{1500}$$

$$CR(A_1) = \frac{500 \times n}{1500} \Rightarrow \infty$$

Online Algorithms: Preformance Measure

Ski Rental: Optimal

$$Opt(1) = 500$$

$$Opt(11) = 1000$$

$$Opt(111) = 1500$$

$$Opt(111\cdots) = 1500$$

Online Algorithms: Preformance Measure

Ski Rental: Optimal

$$Opt(1) = 500$$

$$Opt(11) = 1000$$

$$Opt(111) = 1500$$

$$Opt(111\cdots) = 1500$$

Ski Rental: Algorithm 2

$A_2(1) = 1500, A_2(11) = 1500, A_2(111) = 1500, A_2(1111\cdots) = 1500$

$$\frac{A_2(1)}{OPT(1)} = \frac{1500}{500}, \frac{A_2(11)}{OPT(11)} = \frac{1500}{1000}, \frac{A_2(111)}{OPT(111)} = \frac{1500}{1500}, \frac{A_2(1111)}{OPT(1111)} = \frac{1500}{1500}$$

$$CR(A_2) = \frac{1500}{500} = 3$$

Online Algorithms: Preformance Measure

Ski Rental: Optimal

$$Opt(1) = 500$$

$$Opt(11) = 1000$$

$$Opt(111) = 1500$$

$$Opt(111\cdots) = 1500$$

Online Algorithms: Preformance Measure

Ski Rental: Optimal

$$Opt(1) = 500$$

$$Opt(11) = 1000$$

$$Opt(111) = 1500$$

$$Opt(111\cdots) = 1500$$

Ski Rental: Algorithm 3

$A_3(1) = 500, A_3(11) = 1000, A_3(111) = 2500, A_3(1111\cdots) = 2500$

$$\frac{A_3(1)}{OPT(1)} = \frac{500}{500}, \frac{A_3(11)}{OPT(11)} = \frac{1000}{1000}, \frac{A_3(111)}{OPT(111)} = \frac{2500}{1500}, \frac{A_3(1111)}{OPT(1111)} = \frac{2500}{1500}$$

$$CR(A_2) = \frac{2500}{1500} = 1.67$$

Online Algorithms: Preformance Measure

Comptitive Ratio

$$CR(A) = \max_i \frac{A(i)}{Opt(i)}$$

Optimal is known in Ski Rental Problem.

Online Algorithms: Preformance Measure

Comptitive Ratio

$$CR(A) = \max_i \frac{A(i)}{Opt(i)}$$

Optimal is known in Ski Rental Problem.

Online Algorithms: Preformance Measure

Comptitive Ratio

$$CR(A) = \max_i \frac{A(i)}{Opt(i)}$$

Optimal is known in Ski Rental Problem.
There are some problems for which we do not know the optimal solution.

Online Algorithms: Preformance Measure

Comptitive Ratio

$$CR(A) = \max_i \frac{A(i)}{Opt(i)}$$

Optimal is known in Ski Rental Problem.
There are some problems for which we do not know the optimal solution.
How to compute competitive ratio?

Online Algorithms

Linear List Search

Given a set of items in a list where the cost of accessing an item is proportional to its distance from the head of the list, e.g. a Linked List, and a request sequence of accesses, the problem is to come up with a strategy of reordering the list so that the total cost of accesses is minimized.

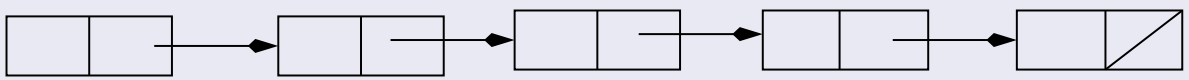


Figure: Linear Linked List.

Online Algorithms

Linear List Search

Input: Initial list $l = (1, 2, 3, \cdots, n)$ and sequence of requests $i_1, i_2, \cdots$

Action: Search item i_j

Goal: minimize the total cost.

Stretegy: move the access item forward.

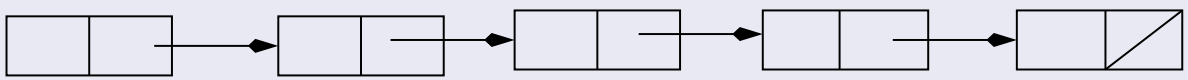


Figure: Linear Linked List.

Question: How many positions?

Online Algorithms

Linear List Search

Stretegy: move the access item forward.

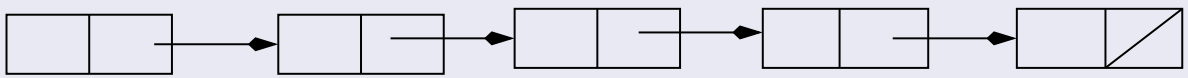


Figure: Linear Linked List.

Question: How many positions?
There are many possible choices: to the top, by 2 positions, to the middle ect.
If you move to the top, then in the worst case you never access that element.

Online Algorithms

Linear List Search

Stretegy: move the access item forward by 2 position.

Online Algorithms

Linear List Search

Stretegy: move the access item forward by 2 position.

Online Algorithms

Linear List Search

Stretegy: move the access item forward by 2 position.
“2 position is not good”

Online Algorithms

Linear List Search

Stretegy: move the access item forward by 2 position.
"2 position is not good"
List: $1, 2, 3 \cdots n$
Requests: $n, n, n \cdots$ (n times)
Costs: $n + (n-2) + \cdots = n^2$
Opt: $n + 1 + 1 + \cdots$ = 2n
$CR = n$
$CR \Rightarrow \infty$ if $n \Rightarrow \infty$

Linear List Search

Move To Front (MTF)

After accessing an item move it to the front of the list without changing the order of other items.

Linear List Search

Move To Front (MTF)

After accessing an item move it to the front of the list without changing the order of other items.

Linear List Search

Move To Front (MTF)

After accessing an item move it to the front of the list without changing the order of other items.

Theorem

MTF has a CR of 2.0

Linear List Search

Theorem

MTF has a CR of 2.0

Linear List Search

Theorem

MTF has a CR of 2.0

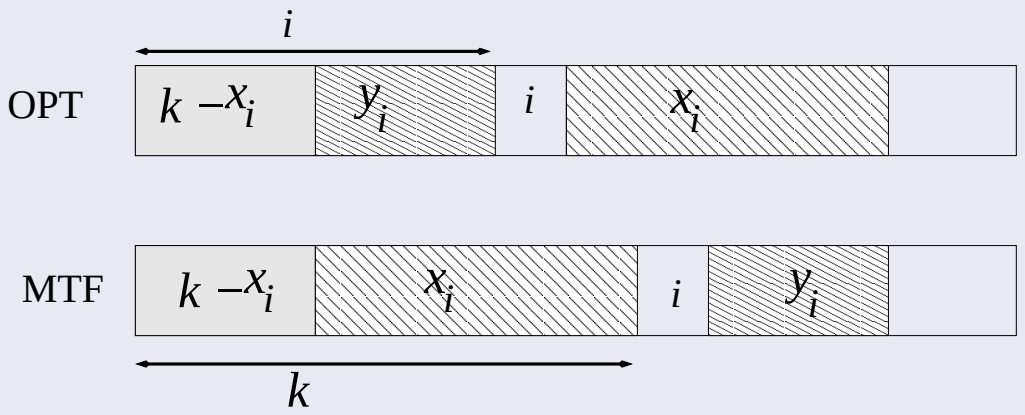
Proof.

We do not know the optimal algorithm, but we need to compute CR.
We compare two algorithms: OPT (which we do not know) and MTF.
At the begining:
OPT: $1, 2, 3, \cdots, n$
MFT: $1, 2, 3, \cdots, n$ □

Linear List Search

Proof.

At Step j:



The cost of MTF is k.

x_i: number of items which are after i in OPT list and before i in MTF list

y_i: number of items which are before i in OPT list and after i in MTF list

Linear List Search

Proof.

Amortized Cost

$$T(j) = k + INV(j+1) - INV(j)$$

$INV(j)$: number of item pairs (a, b) such that the order of a and b is different between the OPT and the MTF lists at time j.

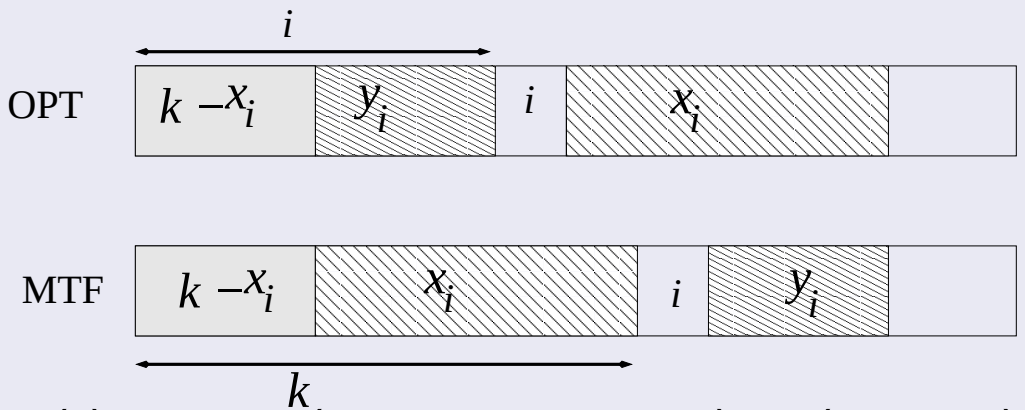
OPT: $\cdots a \cdots b \cdots$

MTF: $\cdots b \cdots a \cdots$

Linear List Search

Proof.

At Step j:



$$\begin{aligned} T(j) &= k + (k - x_i + y_i - \alpha) - (x_i + y_i) \\ &= 2k - 2x_i - \alpha \\ &\leq 2k - 2x_i \\ &\leq 2i \end{aligned}$$

Linear List Search

Proof.

$$T(1) = k_1 + INV(2) - INV(1)$$
$$T(2) = k_2 + INV(3) - INV(2)$$
$$\vdots$$
$$T(m) = k_m + INV(m+1) - INV(m)$$

$$T(1) + T(2) + \cdots T(m) = k_1 + \cdots + k_m + INV(m+1) - INV(1)$$

Linear List Search

Proof.

$$T(1) + T(2) + \cdots T(m) = k_1 + \cdots + k_m + INV(m+1) - INV(1)$$

$$\begin{aligned} k_1 + k_2 + \cdots + k_m &\leq T(1) + T(2) + \cdots T(m) - INV(m+1) \\ &\leq T(1) + T(2) + \cdots T(m) \\ &\leq 2OPT \end{aligned}$$