Lecture 6.
Online Algorithms

Introduction to Algorithms

Da Nang University of Science and Technology

Dang Thien Binh

dtbinh@dut.udn.vn

Contents

- Introduction
- Online algorithms analysis
  - Competitive analysis
  - Probabilistic analysis
- Motivating examples:
  - Ski rental problem
  - 2-lane line search problem
  - Paging problem
  - Bélády's anomaly

Introduction

- An **online algorithm** is one that can process its input piece-by-piece, without having the entire input available from the beginning
  - For example, insertion sort is an online algorithm because:
    - It considers one input element per iteration
    - It produces a partial solution without considering future elements
- In contrast, an **offline algorithm** is given the whole problem data from the beginning
  - For example, selection sort is an offline algorithm because:
    - It requires access to the entire input to sort an array by repeatedly finding the minimum element

Online algorithms analysis

- Competitive analysis
  - An online algorithm is **c-competitive** if there is a constant b for all sequences s of operations

$$C_{ON}(s) \leq c\, C_{OPT}(s) + b$$

Where $C_{ON}(s)$ is the cost of online algorithm on the sequence s and $C_{OPT}(s)$ is the optimal offline cost for the same sequence

  - Competitive ratio: The ratio between the output generated by an online algorithm and the output produced by an optimal offline algorithm
- Probabilistic analysis
  - Assume a distribution generating the input then find an algorithm which minimizes the expected cost of the online algorithm

Ski rental problem (1/4)

- Problem description: We will go skiing in Pyeongchang for the first time. We have two options for the equipment: (1) Buying it costs $500; (2) Renting it for a weekend costs $50. Should we buy or rent it?
- Our goal is to minimize the cost for equipment
- Observations: The cost depends on how many times we will go skiing. Specifically,
  - If we will go skiing 11 times or more, then it is better to buy the equipment
  - If we will go skiing 9 times or fewer, then it is better to rent the equipment

Ski rental problem (2/4)

- An online algorithm called ***better-late-than-never***:
  - rent until you realize you should have bought, then buy. (In our case: rent 9 times, then buy).
  - will be a number k such that after renting k-1 times, we will buy the equipment (just before your k^{th} time)
  - Formally, if the rental cost is r and the purchase cost is p then the algorithm is to rent $\lceil p/r \rceil - 1$ times and then buy
- Competitive analysis:
  - For k < 10, we should rent -> the cost for optimal offline algorithm is **50k**, so the competitive ratio is

$$\frac{50k}{50k} = 1$$

  - If k ≥ 10, we should buy -> the cost for optimal offline algorithm is **500**, so the competitive ratio is

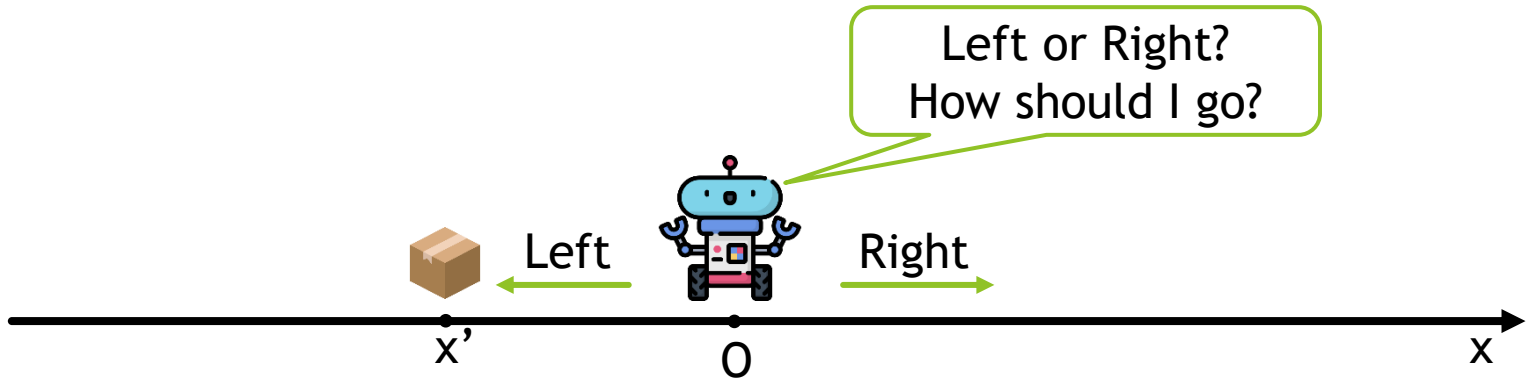
$$\frac{r\left(\left\lceil \frac{p}{r} \right\rceil - 1\right) + p}{p} = \left(2 - \frac{r}{p}\right) \leq 2$$

Ski rental problem (4/4)

- Conclusion: The algorithm ***better-late-than-never*** has competitive ratio ≤ **2**. If the purchase cost ***p*** is a multiple of rental cost ***r***, then the competitive ratio is **2 - r/p**
  - So ***better-late-than-never*** algorithm is ***(2-r/p)-competitive***

2-lane line search problem (1/5)

- Problem description: A robot starts at the origin of the x-axis. An object has been placed somewhere on the x-axis. The robot can switch direction of travel immediately, but in order for the robot to determine that there is an object at location x', the robot has to be physically present at x'. How should the robot explore the x-axis in order to find the object as soon as possible?



2-lane line search problem (2/5)

- We investigate an online algorithm called ***doubling strategy*** to find the object

Input:

1) *Current direction:= Left*
2) *Current position:= 0*
3) *d:= 1*

Repeat:

1) *Travel d unit on the current direction*
2) *If found the object then finish*
3) *Else return to starting point*
4) *d:= 2d*
5) *Flip the current direction*

2-lane line search problem (3/5)

- The robot follows an online algorithm named ***doubling strategy*** to find the object

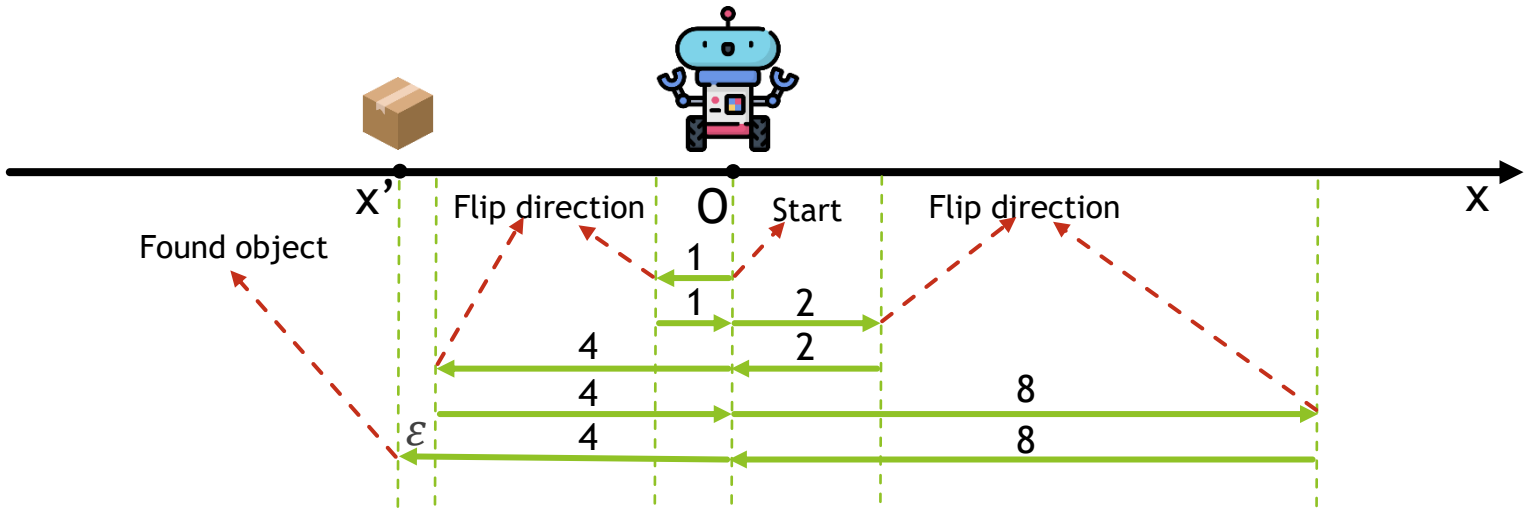


Figure 3. The worst case when the robot finds the object

2-lane line search problem (4/5)

- We derive the competitive ratio of the ***doubling strategy***
- If the object locates at position $x' = 2^j + \varepsilon$.
- We have the total distance of this algorithm:

$$Cost_{ON} = 2\underbrace{(1 + 2 + 4 + \ldots + 2^{j+1})}_{} + 2^j + \varepsilon$$

$$= 2 * \frac{1 * (1 - 2^{j+2})}{1 - 2} + 2^j + \varepsilon$$

$$= 2 * (2^{j+2} - 1) + 2^j + \varepsilon$$

$$= (2^3 + 1) * 2^j + \varepsilon - 2$$

$$\approx 9 * 2^j + \varepsilon$$

where j is the number of iterations and ε is a small distance

2-lane line search problem (5/5)

- The cost of optimal offline algorithm when the robot knows where the object locates at $x' = 2^j + \varepsilon$

$$Cost_{OPT} = x' = 2^j + \varepsilon$$

- Finally, combine $Cost_{ON}$ and $Cost_{OPT}$, we obtain the competitive ratio as following

$$\text{Competitive ratio} = \frac{Cost_{ON}}{Cost_{OPT}} = \frac{9*2^j+\varepsilon}{2^j+\varepsilon} \approx 9$$

- Conclusion: The doubling strategy is 9-competitive for the 2-lane line search problem

Paging problem (1/5)

- Problem description: We have a disk with N pages, and cache with space k < N. When a memory request is made, if the page isn't in the cache -> page fault. We then need to bring the page into the cache and throw something else out if our cache is full. Our goal is to minimize the number of page faults. The algorithmic question is: which page should we throw out?
- There are some online algorithms for this problem:
  - **Least-Recently-Used (LRU)** (We will investigate this algorithm)
  - First-In-First-Out (FIFO)
  - Flush-When-Full (FWF)

3 phương pháp

Paging problem (2/5)

- Least-Recently-Used (LRU) algorithm evicts the page that has been unused for the longest time

- Competitive ratio is used to analyze LRU algorithm

$$\text{Competitive ratio} = \frac{\#\ of\ evictions\ of\ online\ algorithm(\sigma)}{\#\ of\ evictions\ of\ optimal\ offline\ algorithm(\sigma)}$$

where σ is the input sequence of requests

Paging problem (3/5)

- We will derive the competitive ratio of LRU algorithm
- Consider the cache has k slots, which are initially empty and the input sequence

$$\sigma = 1, 2, 3, \ldots, k, k+1, 1, 2, 3, \ldots, k$$

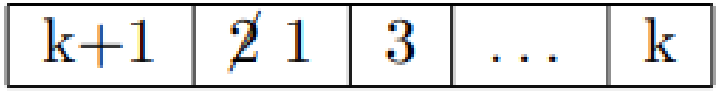
  - After arriving of first k inputs, cache looks like

  - When k+1 arrives, 1 will be evicted because 1 is unused for the longest time, cache then looks like

  - Continue…

Paging problem (4/5)

- When 1 arrives again, 2 will be evicted, cache looks like

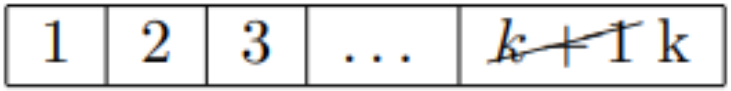


- Similarly, all the k pages will be evicted one by one

- And, when the last input k arrives, k+1 will be evicted

Paging problem (5/5)

- So, the total number of evictions using LRU is k+1
- Now, if we use the optimal algorithm for the same input sequence σ, the cache will look like



  - Hence, the number of evictions for optimal algorithm is 2

$$\text{Competitive ratio} = \frac{k+1}{2}$$

Bélády's anomaly (1/5)

- Bélády's anomaly is a phenomenon where **increasing** the size of cache memory results in an **increase** of page faults
- Bélády's anomaly is commonly experienced with FIFO algorithm but never occurs in LRU algorithm
- Reason behind Bélády's anomaly:
  - LRU is a stack-based algorithm -> Do not suffer Bélády's anomaly
  - Stack property: the set of pages that were presented in cache memory when the size of cache memory is k will be compulsorily presented in cache memory if the size of cache memory increases to k’>k

Bélády's anomaly (2/5)

Một số trường hợp sẽ bị

- FIFO algorithm example:
  - Given the request: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- Case 1: If **k = 3**, the given request using FIFO algorithm yields a total of **9** page faults (PF)

Stage 1	Stage 2	Stage 3	Stage 4	Stage 5	Stage 6	Stage 7	Stage 8	Stage 9	Stage 10	Stage 11	Stage 12
(1)	1	1	2	3	4	1	(1)	1	2	5	(5)
	(2)	2	3	4	1	2	2	(2)	5	3	3
		(3)	(4)	(1)	(2)	(5)	5	5	(3)	(4)	4
PF	PF	PF	PF	PF	PF	PF	0	0	PF	PF	0

- Case 2: If **k = 4**, the given request using FIFO algorithm yields a total of **10** page faults (PF)

Stage 1	Stage 2	Stage 3	Stage 4	Stage 5	Stage 6	Stage 7	Stage 8	Stage 9	Stage 10	Stage 11	Stage 12
(1)	1	1	1	(1)	1	2	3	4	5	1	2
	(2)	2	2	2	(2)	3	4	5	1	2	3
		(3)	3	3	3	4	5	1	2	3	4
			(4)	4	4	(5)	(1)	(2)	(3)	(4)	(5)
PF	PF	PF	PF	0	0	PF	PF	PF	PF	PF	PF

Bélády's anomaly (3/5)

- At stage 7 and stage 8 in Case 2, cache memory does not contain the set of pages that are presented in the corresponding stages in Case 1
- Thus, FIFO algorithm does not follow the stack property
- FIFO algorithm suffers Bélády's anomaly
  - The number of page faults increases from 9 to 10 when increasing the size of cache memory from 3 to 4
- **Note**: It would be wrong to say that “FIFO algorithm always suffers from Bélády's anomaly”

Bélády's anomaly (4/5)

- LRU algorithm example:
  - Given the request: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- If **k = 3**, the given request using LRU algorithm yields a total of **10** page faults (PF)

Stage 1	Stage 2	Stage 3	Stage 4	Stage 5	Stage 6	Stage 7	Stage 8	Stage 9	Stage 10	Stage 11	Stage 12
(1)	(2)	(3)	(4)	(1)	(2)	(5)	(1)	(2)	(3)	(4)	(5)
	1	2	3	4	1	2	5	1	2	3	4
		1	2	3	4	1	2	5	1	2	3
PF	PF	PF	PF	PF	PF	PF	0	0	PF	PF	PF

- If **k = 4**, the given request using LRU algorithm yields a total of **8** page faults (PF)

Stage 1	Stage 2	Stage 3	Stage 4	Stage 5	Stage 6	Stage 7	Stage 8	Stage 9	Stage 10	Stage 11	Stage 12
(1)	(2)	(3)	(4)	(1)	(2)	(5)	(1)	(2)	(3)	(4)	(5)
	1	2	3	4	1	2	5	1	2	3	4
		1	2	3	4	1	2	5	1	2	3
			1	2	3	4	4	4	5	1	2
PF	PF	PF	PF	0	0	PF	0	0	PF	PF	PF

Bélády's anomaly (5/5)

- At stages in Case 2, cache memory contains the set of pages that are presented in the corresponding stages in Case 1
- LRU algorithm follows the stack property
- LRU does not suffer Bélády's anomaly:
  - The number of page faults decreases from 10 to 8 when increasing the size of cache memory from 3 to 4

Thanks to contributors

Mr. Phuoc-Nguyen Bui (2022)

Dr. Thien-Binh Dang (2017 – 2022)

Prof. Hyunseung Choo (2001 – 2022)