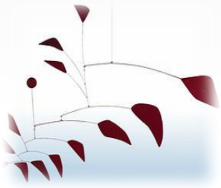


EECS 4101-5101
Advanced Data Structures

Shahin Kamali

Topic 1c - Competitive Analysis

CLRS 17-1, 17-2, 17-3, 17-4

York University

Picture is from the cover of the textbook CLRS.

Offline vs. Online Algorithms

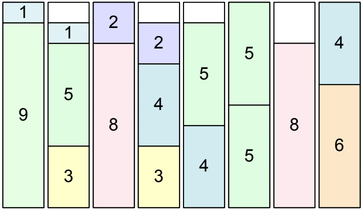
- Traditional algorithms are ‘offline’ in the sense that they have the whole input in their hand.
- Online algorithms, in contrast, do not have/need the whole input in order to solve a problem
  - The input is a ‘sequence’ which is processed by the online algorithm **piece-by-piece**
  - The online algorithms often take **irrevocable decisions** to process the input.

Bin Packing Problem

- The input is a set/sequence of items of various sizes
  - E.g., $< 9, 3, 8, 5, 1, 1, 3, 2, 4, 2, 4, 5, 5, 8, 6, 4, 5, ... >$.

Bin Packing Problem

- The input is a set/sequence of items of various sizes
  - E.g., $< 9, 3, 8, 5, 1, 1, 3, 2, 4, 2, 4, 5, 5, 8, 6, 4, 5, \ldots >$.
- The goal is to pack these items into a minimum number of bins of uniform capacity.



Bin Packing Problem (cntd.)

- In the online setting:
  - an algorithm receives items one by one
  - when it receives an item, it has to place it in a bin without any knowledge about forthcoming items
  - decisions of the algorithms are irrevocable (i.e., cannot move items between bins)

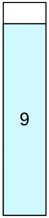
First Fit (FF) Algorithm

- Find the first bin which has enough space for the item, and place the item there
- Open a new bin if such bin does not exist

First Fit (FF) Algorithm

- Find the first bin which has enough space for the item, and place the item there
- Open a new bin if such bin does not exist

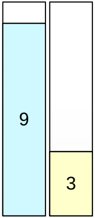
< 9 3 8 5 1 1 3 2 4 2 4 5 5 8 6 4 5 >



First Fit (FF) Algorithm

- Find the first bin which has enough space for the item, and place the item there
- Open a new bin if such bin does not exist

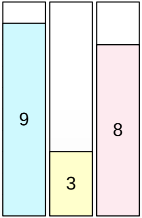
$$< 9\ 3\ 8\ 5\ 1\ 1\ 3\ 2\ 4\ 2\ 4\ 5\ 5\ 8\ 6\ 4\ 5 >$$



First Fit (FF) Algorithm

- Find the first bin which has enough space for the item, and place the item there
- Open a new bin if such bin does not exist

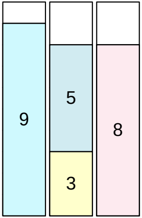
$$< 9\ 3\ 8\ 5\ 1\ 1\ 3\ 2\ 4\ 2\ 4\ 5\ 5\ 8\ 6\ 4\ 5 >$$



First Fit (FF) Algorithm

- Find the first bin which has enough space for the item, and place the item there
- Open a new bin if such bin does not exist

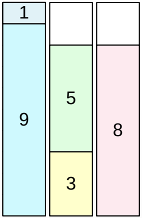
$$< 9\ 3\ 8\ 5\ 1\ 1\ 3\ 2\ 4\ 2\ 4\ 5\ 5\ 8\ 6\ 4\ 5 >$$



First Fit (FF) Algorithm

- Find the first bin which has enough space for the item, and place the item there
- Open a new bin if such bin does not exist

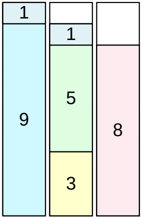
$$< 9\ 3\ 8\ 5\ \color{red}{1}\ 1\ 3\ 2\ 4\ 2\ 4\ 5\ 5\ 8\ 6\ 4\ 5 >$$



First Fit (FF) Algorithm

- Find the first bin which has enough space for the item, and place the item there
- Open a new bin if such bin does not exist

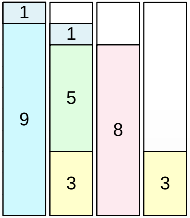
$$< 9\ 3\ 8\ 5\ 1\ \color{red}{1}\ 3\ 2\ 4\ 2\ 4\ 5\ 5\ 8\ 6\ 4\ 5 >$$



First Fit (FF) Algorithm

- Find the first bin which has enough space for the item, and place the item there
- Open a new bin if such bin does not exist

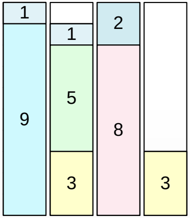
$$< 9\ 3\ 8\ 5\ 1\ 1\ {\color{red}3}\ 2\ 4\ 2\ 4\ 5\ 5\ 8\ 6\ 4\ 5 >$$



First Fit (FF) Algorithm

- Find the first bin which has enough space for the item, and place the item there
- Open a new bin if such bin does not exist

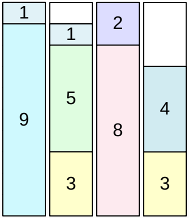
$$< 9\ 3\ 8\ 5\ 1\ 1\ 3\ \color{red}{2}\ 4\ 2\ 4\ 5\ 5\ 8\ 6\ 4\ 5 >$$



First Fit (FF) Algorithm

- Find the first bin which has enough space for the item, and place the item there
- Open a new bin if such bin does not exist

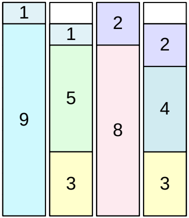
$$< 9\ 3\ 8\ 5\ 1\ 1\ 3\ 2\ 4\ 2\ 4\ 5\ 5\ 8\ 6\ 4\ 5 >$$



First Fit (FF) Algorithm

- Find the first bin which has enough space for the item, and place the item there
- Open a new bin if such bin does not exist

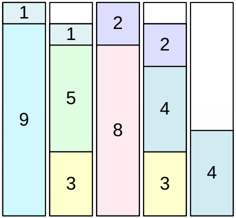
$$< 9\ 3\ 8\ 5\ 1\ 1\ 3\ 2\ 4\ \color{red}{2}\ 4\ 5\ 5\ 8\ 6\ 4\ 5 >$$



First Fit (FF) Algorithm

- Find the first bin which has enough space for the item, and place the item there
- Open a new bin if such bin does not exist

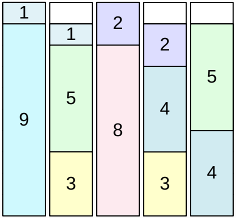
$$< 9\ 3\ 8\ 5\ 1\ 1\ 3\ 2\ 4\ 2\ \color{red}{4}\ 5\ 5\ 8\ 6\ 4\ 5 >$$



First Fit (FF) Algorithm

- Find the first bin which has enough space for the item, and place the item there
- Open a new bin if such bin does not exist

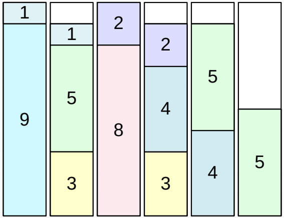
$$< 9\ 3\ 8\ 5\ 1\ 1\ 3\ 2\ 4\ 2\ 4\ 5\ 5\ 8\ 6\ 4\ 5 >$$



First Fit (FF) Algorithm

- Find the first bin which has enough space for the item, and place the item there
- Open a new bin if such bin does not exist

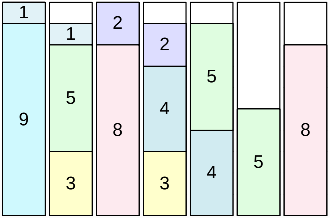
$$< 9\ 3\ 8\ 5\ 1\ 1\ 3\ 2\ 4\ 2\ 4\ 5\ 5\ 8\ 6\ 4\ 5 >$$



First Fit (FF) Algorithm

- Find the first bin which has enough space for the item, and place the item there
- Open a new bin if such bin does not exist

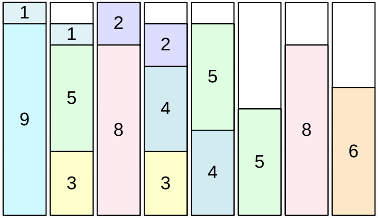
$$< 9\ 3\ 8\ 5\ 1\ 1\ 3\ 2\ 4\ 2\ 4\ 5\ 5\ \color{red}{8}\ 6\ 4\ 5 >$$



First Fit (FF) Algorithm

- Find the first bin which has enough space for the item, and place the item there
- Open a new bin if such bin does not exist

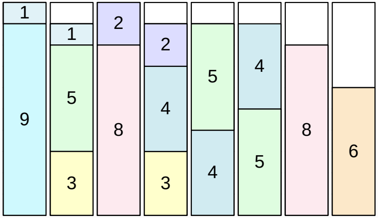
$$< 9\ 3\ 8\ 5\ 1\ 1\ 3\ 2\ 4\ 2\ 4\ 5\ 5\ 8\ {\color{red}6}\ 4\ 5 >$$



First Fit (FF) Algorithm

- Find the first bin which has enough space for the item, and place the item there
- Open a new bin if such bin does not exist

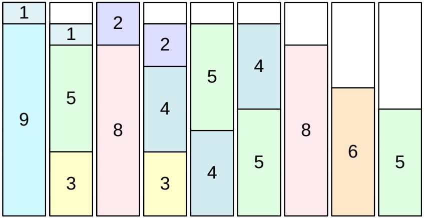
$$< 9\ 3\ 8\ 5\ 1\ 1\ 3\ 2\ 4\ 2\ 4\ 5\ 5\ 8\ 6\ {\color{red}4}\ 5 >$$



First Fit (FF) Algorithm

- Find the first bin which has enough space for the item, and place the item there
- Open a new bin if such bin does not exist

$$< 9\ 3\ 8\ 5\ 1\ 1\ 3\ 2\ 4\ 2\ 4\ 5\ 5\ 8\ 6\ 4\ \color{red}{5}\ >$$



Competitive Ratio

- We use the framework of **competitive analysis** to compare online algorithms

Competitive Ratio

- We use the framework of **competitive analysis** to compare online algorithms
- Let Opt denote the best possible offline solution.
  - Given a sequence σ, Opt is an algorithm which packs items in σ in a minimum number of bins

Competitive Ratio

- We use the framework of **competitive analysis** to compare online algorithms
- Let Opt denote the best possible offline solution.
  - Given a sequence σ, Opt is an algorithm which packs items in σ in a minimum number of bins
- Competitive ratio of an algorithm A is the maximum ratio between the cost of A and that of Opt over all sequences

$$cr(A) \equiv \max_{\sigma} \frac{cost_A(\sigma)}{cost_{\mathrm{Opt}}(\sigma)}$$

Competitive Ratio of First Fit

- For First Fit, the competitive ratio is 1.7 [Johnson 1973]

Competitive Ratio of First Fit

- For First Fit, the competitive ratio is 1.7 [Johnson 1973]
  - The number of bins opened by FF for **any** sequence is at most 1.7 times that of Opt, i.e., $c.r. \leq 1.7$ (upper bound for FF)
  - There are sequences for which the number of bins opened by FF is 1.7 times that of Opt, i.e., $c.r. \geq 1.7$ (lower bound for FF)

Competitive Ratio of First Fit

- For First Fit, the competitive ratio is 1.7 [Johnson 1973]
  - The number of bins opened by FF for **any** sequence is at most 1.7 times that of Opt, i.e., $c.r. \leq 1.7$ (upper bound for FF)
  - There are sequences for which the number of bins opened by FF is 1.7 times that of Opt, i.e., $c.r. \geq 1.7$ (lower bound for FF)
- The best existing online algorithm has c.r. of 1.5783 [Balogh et al. 2017]

Competitive Ratio of First Fit

- For First Fit, the competitive ratio is 1.7 [Johnson 1973]
  - The number of bins opened by FF for **any** sequence is at most 1.7 times that of Opt, i.e., $c.r. \leq 1.7$ (upper bound for FF)
  - There are sequences for which the number of bins opened by FF is 1.7 times that of Opt, i.e., $c.r. \geq 1.7$ (lower bound for FF)
- The best existing online algorithm has c.r. of 1.5783 [Balogh et al. 2017]
- No algorithm can be better than 1.54037-competitive (best general lower bound) [Balogh et al. 2015].

Ski-rental problem

- Assume you want to go skiing for x number of days
  - In the online setting, the value of x is unknown!

Ski-rental problem

- Assume you want to go skiing for x number of days
  - In the online setting, the value of x is unknown!
- You can buy the equipment for a one-time cost of b or rent each day for a cost of 1 per day

Ski-rental problem

- Assume you want to go skiing for x number of days
  - In the online setting, the value of x is unknown!
- You can buy the equipment for a one-time cost of b or rent each day for a cost of 1 per day
- If we know x, what is the best solution?

Ski-rental problem

- Assume you want to go skiing for x number of days
  - In the online setting, the value of x is unknown!
- You can buy the equipment for a one-time cost of b or rent each day for a cost of 1 per day
- If we know x, what is the best solution?
  - Buy at the beginning if $x \geq b$, otherwise, rent every day

Ski-rental problem

- Assume you want to go skiing for x number of days
  - In the online setting, the value of x is unknown!
- You can buy the equipment for a one-time cost of b or rent each day for a cost of 1 per day
- If we know x, what is the best solution?
  - Buy at the beginning if $x \geq b$, otherwise, rent every day
- What is the competitive ratio of an algorithm that buys at day 1?

Ski-rental problem

- Assume you want to go skiing for x number of days
  - In the online setting, the value of x is unknown!
- You can buy the equipment for a one-time cost of b or rent each day for a cost of 1 per day
- If we know x, what is the best solution?
  - Buy at the beginning if $x \geq b$, otherwise, rent every day
- What is the competitive ratio of an algorithm that buys at day 1?
  - In the worst case, you go skiing once; so $\frac{b}{1} = b$ (not good)

Ski-rental problem

- Assume you want to go skiing for x number of days
  - In the online setting, the value of x is unknown!
- You can buy the equipment for a one-time cost of b or rent each day for a cost of 1 per day
- If we know x, what is the best solution?
  - Buy at the beginning if $x \geq b$, otherwise, rent every day
- What is the competitive ratio of an algorithm that buys at day 1?
  - In the worst case, you go skiing once; so $\frac{b}{1} = b$ (not good)
- What is the competitive ratio of an algorithm that always rent?
  - In the worst-case, we go skiing n days for large n
  - The competitive ratio is $\frac{n}{b}$, which can be arbitrary large (very bad).

Ski-rental problem (cntd.)

- Online strategy **break-even**: rent for the first $b-1$ days and buy in the next day.

Ski-rental problem (cntd.)

- Online strategy **break-even**: rent for the first $b-1$ days and buy in the next day.
- What is the competitive ratio of Break-even algorithm?

Ski-rental problem (cntd.)

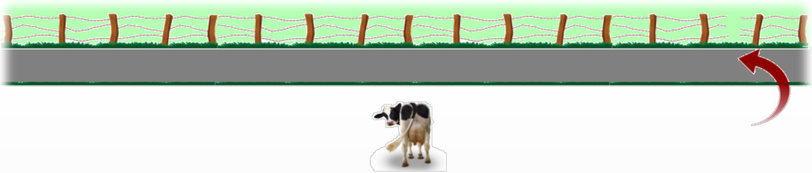
- Online strategy **break-even**: rent for the first $b-1$ days and buy in the next day.
- What is the competitive ratio of Break-even algorithm?
- It is $\frac{(b-1)+b}{b} \approx 2$

> **Theorem**
>
> *Competitive ratio is roughly 2, and it is the best for any* ***deterministic*** *online algorithm.*

Cow-Path Problem

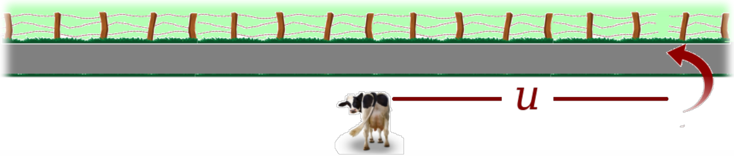
Problem Definition

- A cow faces a fence, infinite in both directions
- She wants to find a hole in order to get to the green pasture on the other side
- The cow's **online strategy** specifies the path traveled in search of the hole.
- The goal is to minimize the distance traveled.



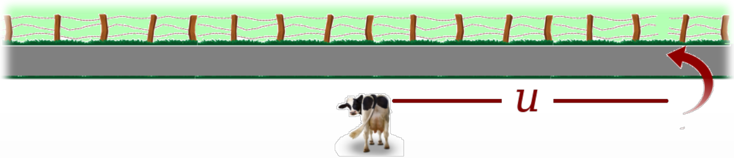
Offline Strategy

- Let u an integer indicating the distance between the initial location of the cow and the location of the hole.
  - u is unknown to the cow!



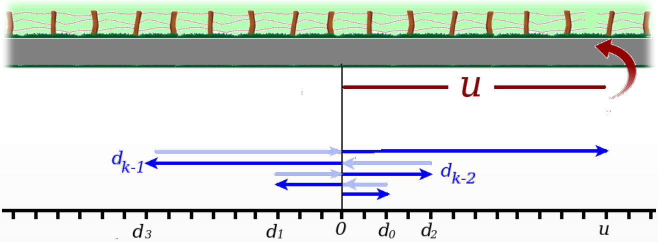
Offline Strategy

- Let u an integer indicating the distance between the initial location of the cow and the location of the hole.
  - u is unknown to the cow!
- An optimal offline algorithm *Opt* (i.e., a cow which knows the location of the hole), incurs a cost of u



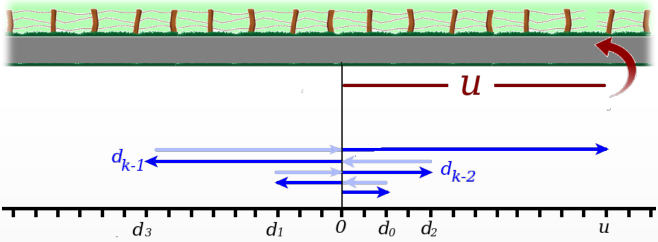
Smart-Cow Algorithm (SCA)

- Gradually extend the explored interval of the fence
- Alternate between left and right!
  - Go right for distance d_0
  - Go back to the origin, left for distance d_1
  - Go back to the origin, right for distance d_2
  - Continue accordingly for $d_3, \ldots, d_k$ until the hole is found.



Competitive Ratio of SCA

- Recall that the competitive ratio of an online algorithm is the maximum ratio between the cost of that algorithm and an optimal offline Opt algorithm Opt
- The cost of Opt is u
- The cost of SCA is $2d_0 + 2d_1 + \ldots + 2d_{k-2} + 2d_{k-1} + u$
    - $d_{k-2} < u \leq d_k$

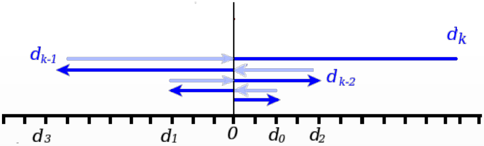


Competitive Ratio of SCA (cntd.)

- The competitive ratio would be

$$\frac{2d_0 + 2d_1 + \ldots + 2d_{k-2} + 2d_{k-1} + u}{u} = 1 + 2\frac{d_0 + d_1 + \ldots + d_{k-1}}{u}$$

- what is the value of u in the worst case?
  - If you are an **adversary** and want to fail the algorithm, where you place the hole?



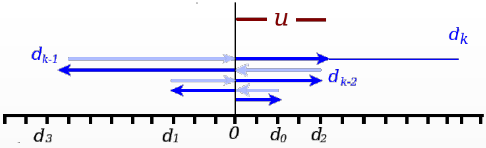
Competitive Ratio of SCA (cntd.)

- The competitive ratio would be

$$\frac{2d_0 + 2d_1 + \ldots + 2d_{k-2} + 2d_{k-1} + u}{u} = 1 + 2\frac{d_0 + d_1 + \ldots + d_{k-1}}{u}$$

- In the worst case, $u = d_{k-2} + \epsilon$.
  - Just a bit more than the previous probe!
- So, the competitive ratio of a Smart-Cow algorithm is

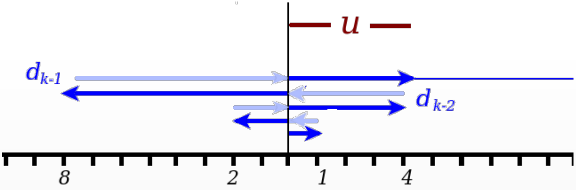
$$1 + 2\frac{d_0 + d_1 + \ldots + d_{k-1}}{d_{k-2} + \epsilon}$$



The Doubling Technique

- Assume $d_i = 2^i$, i.e., first go one unit to the right, go back to the origin, go two units to the left, back to origin, four units to the right, etc.
  - We will have $d_0 + d_1 + \ldots + d_{k-1} = 1 + 2 + 4 + \ldots + 2^{k-1} = \cdot 2^k - 1 = 4 \cdot 2^{k-2}$.
  - The competitive ratio would be

$$1 + 2\frac{d_0 + d_1 + \ldots + d_{k-1}}{d_{k-2} + \epsilon} = 1 + 2\frac{4 \cdot 2^{k-2}}{2^{k-2} + \epsilon} \approx 9$$



Overview

Theorem

The smart-cow algorithm with steps that double (i.e., $d_i = 2^i$) has a competitive ratio of at most 9.

Overview

Theorem

The smart-cow algorithm with steps that double (i.e., $d_i = 2^i$) has a competitive ratio of at most 9.

- It turns out that no **deterministic** algorithm can achieve a ratio better than 9.
  - The proof is a bit involved and we skip it here.

Overview

Theorem

The smart-cow algorithm with steps that double (i.e., $d_i = 2^i$) has a competitive ratio of at most 9.

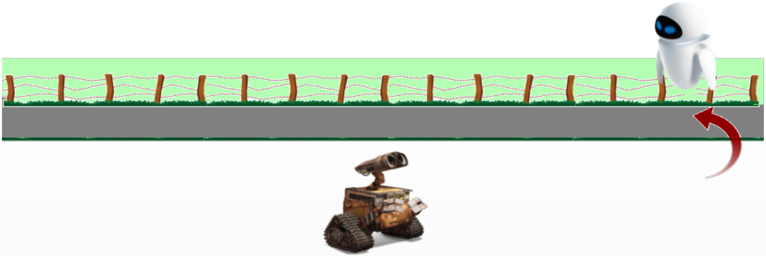
- It turns out that no **deterministic** algorithm can achieve a ratio better than 9.
  - The proof is a bit involved and we skip it here.
- So, the doubling technique results an optimal algorithm in this case

Semi-online Problem

- We assumed the value of u is unknown to the algorithm.
- **Question:** what competitive an "almost-online" algorithm can achieve when the value of u is known?
  - The algorithm knows u but does not know the side (left or right) where the target is located.

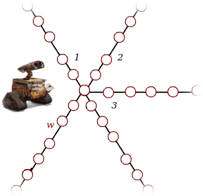
Search Problems under Uncertainty

- A cow can be a robot (or the other way around)!
- In practice, robots often do not have full information about their environment.
- Cow-path problem and its variant are a way to model many types of **search problems**.



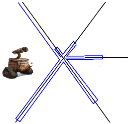
Variants of Search Problems

- Path-cow problem is an online search problem on a **path**.
- Consider a **star**, where w paths have one common endpoint.
- Assume a robot is initially locate at the common point, and needs to find a **target** located in an unknown position.
- What is a good algorithm?



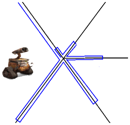
Variants of Search Problems

- The best strategy is to have $d_i = (w/(w-1))^i$.
  - For $w = 2$, it requires doubling.
  - For $w = 3$, we jump by a factor of $3/2$, and so on.



Variants of Search Problems

- The best strategy is to have $d_i = (w/(w-1))^i$.
  - For $w = 2$, it requires doubling.
  - For $w = 3$, we jump by a factor of $3/2$, and so on.
- The competitive ratio will be at most $1 + 2\frac{w^w}{(w-1)^{w-1}} \approx 1 + 2e(w-1)$ (when w is sufficiently large).
  - $e \approx 2.71$ is the Euler's constant



Variants of Search Problems

- The best strategy is to have $d_i = (w/(w-1))^i$.
  - For $w = 2$, it requires doubling.
  - For $w = 3$, we jump by a factor of $3/2$, and so on.
- The competitive ratio will be at most $1 + 2\frac{w^w}{(w-1)^{w-1}} \approx 1 + 2e(w-1)$ (when w is sufficiently large).
  - $e \approx 2.71$ is the Euler's constant
- Note that doubling is not optimal here.
  - But it is still **competitive**, i.e., it has a constant competitive ratio.