COMP 7720 - Online Algorithms

Paging and \( k \)-Server Problem

Shahin Kamali

Lecture 9 - Oct. 4, 2018

University of Manitoba
Review & Plan

Review & Plan
Today’s objectives

- Caching problem & advice
- $k$-server problem
  - Introduction
  - Greedy algorithms
Caching Problem
Problem Definition

There are two types of memory: a fast ‘cache’ of size $k$, and a slow memory of unbounded size.

- The input is an online sequence of requests to pages of size 1.

To serve a request to page $x$, it should be in the cache

- In case $x$ is not in the cache, a fault of cost 1 happens
- In case $x$ is in the cache, a hit of cost 0 happens
- The goal is to minimize the total number of faults

To bring $x$ to the cache, we might need to evict a page.

- A caching algorithm is defined through its eviction policy.
Caching problem: a review

- Latest-In-Future (LIF) is the optimal offline algorithm.
- No deterministic algorithm has a competitive ratio better than $k$. 
Caching Problem

Caching problem: a review

- Latest-In-Future (LIF) is the optimal offline algorithm.
- No deterministic algorithm has a competitive ratio better than $k$.
- An algorithm is marking if it maintains a ‘mark’ for each page.
  - After a request to $x$ mark it.
  - Always evict an unmarked page (if all marked, first unmark all pages and then evict one)
Latest-In-Future (LIF) is the optimal offline algorithm.

No deterministic algorithm has a competitive ratio better than $k$.

An algorithm is marking if it maintains a ‘mark’ for each page.

- After a request to $x$ mark it.
- Always evict an unmarked page (if all marked, first unmark all pages and then evict one)

Any deterministic marking algorithm has a competitive ratio of $k$.

- Least-Recently-Used (LRU), and Flash-When-Full (FWF) both have competitive ratio $k$. 
Caching Problem

Caching problem: a review

- Latest-In-Future (LIF) is the optimal offline algorithm.
- No deterministic algorithm has a competitive ratio better than $k$.
- An algorithm is marking if it maintains a ‘mark’ for each page.
  - After a request to $x$ mark it.
  - Always evict an unmarked page (if all marked, first unmark all pages and then evict one)
- Any deterministic marking algorithm has a competitive ratio of $k$.
  - Least-Recently-Used (LRU), and Flash-When-Full (FWF) both have competitive ratio $k$.
- Fist-In-First-Out (FIFO) also has a competitive ratio of $k$. 
A randomized algorithm which randomly evict a page has a competitive ratio of $k$.

A marking algorithm that evicts an unmarked page uniformly at random has a competitive ratio of $H_k$

- $H_k = 1 + 1/2 + 1/3 + \ldots + 1/k$
- For large values of $k$, we have $H_k \approx \ln(k) \in \Theta(\log k)$.

In fact, no randomized algorithm can achieve a better competitive ratio (i.e., $o(\log k)$)
A randomized algorithm which randomly evicts a page has a competitive ratio of $k$.

A marking algorithm that evicts an unmarked page uniformly at random has a competitive ratio of $H_k$

- $H_k = 1 + 1/2 + 1/3 + \ldots + 1/k$
- For large values of $k$, we have $H_k \approx \ln(k) \in \Theta(\log k)$.

In fact, no randomized algorithm can achieve a better competitive ratio (i.e., $o(\log k)$).
How many bits of advice are sufficient to achieve an optimal algorithm?

- \(n\): the length of input sequence (number of requests).
- \(k\): the size of the cache
- Hint: an algorithm has to make at most \(n\) decisions about the page to be evicted.
  - one decision per fault.

At most \(O(n \log k)\) bits are sufficient!
How many bits of advice are sufficient to achieve an optimal algorithm?

- $n$: the length of input sequence (number of requests).
- $k$: the size of the cache
- Hint: an algorithm has to make at most $n$ decisions about the page to be evicted.
  - one decision per fault.

At most $O(n \log k)$ bits are sufficient!
Caching & advice

- What does the advice encode?
- What is the size of advice?
- How the algorithm works, provided by these bits of advice?
- Why the algorithm has a competitive ratio of 1 (optimal here)?
Caching Problem

Caching & advice

- What does the advice encode? The advice indicates, for each request, what an optimal offline algorithm Opt evicts in case of a failure.
- What is the size of advice?
- How the algorithm works, provided by these bits of advice?
- Why the algorithm has a competitive ratio of 1 (optimal here)?
Caching Problem

Caching & advice

What does the advice encode? The advice indicates, for each request, what an optimal offline algorithm $Opt$ evicts in case of a failure.

What is the size of advice? Assume $Opt$ makes $m \leq n$ faults for the optimal algorithm. For each fault, the advice indicates which page should be evicted. There are $k$ pages in the cache, and the evicted page can be indicated in $\Theta(\log k)$ bits. The total number of bits will be $m \cdot O(\log k) \in O(n \log k)$.

How the algorithm works, provided by these bits of advice?

Why the algorithm has a competitive ratio of 1 (optimal here)?
Caching Problem

Caching & advice

- What does the advice encode? The advice indicates, for each request, what an optimal offline algorithm Opt evicts in case of a failure.

- What is the size of advice? Assume Opt makes $m \leq n$ faults for the optimal algorithm. For each fault, the advice indicates which page should be evicted. There are $k$ pages in the cache, and the evicted page can be indicated in $\Theta(\log k)$ bits. The total number of bits will be $m \cdot O(\log k) \in O(n \log k)$.

- How the algorithm works, provided by these bits of advice? It just mimics Opt; whenever there is a fault, it reads the advice to see which page should be evicted.

- Why the algorithm has a competitive ratio of 1 (optimal here)?
Caching Problem

Caching & advice

What does the advice encode? The advice indicates, for each request, what an optimal offline algorithm Opt evicts in case of a failure.

What is the size of advice? Assume Opt makes \( m \leq n \) faults for the optimal algorithm. For each fault, the advice indicates which page should be evicted. There are \( k \) pages in the cache, and the evicted page can be indicated in \( \Theta(\log k) \) bits. The total number of bits will be \( m \cdot O(\log k) \in O(n \log k) \).

How the algorithm works, provided by these bits of advice? It just mimics Opt; whenever there is a fault, it reads the advice to see which page should be evicted.

Why the algorithm has a competitive ratio of 1 (optimal here)? It works exactly like Opt.
Caching Problem

Caching & advice (cntd.)

Theorem

There is an online algorithm that, provided with \( O(n \log k) \) bits of advice, can achieve an optimal solution.

- It is a naive solution :’-)
- Can we achieve an optimal solution with a smaller number of bits of advice?
  - For many problems, the answer is no!
  - For caching problem, we can indeed do better.
Assume $\text{OPT}$ brings a page $x$ to the cache at time $t$.

- Either $\text{OPT}$ evicts $x$ before the next access to $x \rightarrow x$ is mortal.
- $\text{OPT}$ keeps $x$ in the cache until the next access to $x \rightarrow x$ is resident.
Assume \( \text{OPT} \) brings a page \( x \) to the cache at time \( t \).

- Either \( \text{OPT} \) evict \( x \) before the next access to it \( \rightarrow x \) is mortal.
- \( \text{OPT} \) keeps \( x \) in the cache until the next access to \( x \) \( \rightarrow x \) is resident

\[
\sigma = a \quad b \quad c \quad b \quad a \quad d \quad c \quad e \quad f \quad a \quad c \quad d \quad c \quad f \quad a \quad b \quad a \quad e
\]
Assume $\text{OPT}$ brings a page $x$ to the cache at time $t$.

- Either $\text{OPT}$ evict $x$ before the next access to it $\rightarrow x$ is mortal.
- $\text{OPT}$ keeps $x$ in the cache until the next access to $x$ $\rightarrow x$ is resident

$a$ is resident

$$\sigma = a\ b\ c\ b\ a\ d\ c\ e\ f\ a\ c\ d\ c\ f\ a\ b\ a\ e$$
Caching Problem

Caching & advice (cntd.)

Assume $\text{OPT}$ brings a page $x$ to the cache at time $t$.

- Either $\text{OPT}$ evict $x$ before the next access to it $\rightarrow x$ is mortal.
- $\text{OPT}$ keeps $x$ in the cache until the next access to $x$ $\rightarrow x$ is resident

$a, b$ are residents

$$\sigma = a \ b \ c \ b \ a \ d \ c \ e \ f \ a \ c \ d \ c \ f \ a \ b \ a \ e$$

| a | b |
Assume \( \text{OPT} \) brings a page \( x \) to the cache at time \( t \).

- Either \( \text{OPT} \) evict \( x \) before the next access to it \( \rightarrow x \) is mortal.
- \( \text{OPT} \) keeps \( x \) in the cache until the next access to \( x \) \( \rightarrow x \) is resident

\[
\sigma = a \ b \ c \ b \ a \ d \ c \ e \ f \ a \ c \ d \ c \ f \ a \ b \ a \ e
\]
Caching Problem

Assume $\text{OPT}$ brings a page $x$ to the cache at time $t$.

- Either $\text{OPT}$ evict $x$ before the next access to it $\rightarrow x$ is mortal.
- $\text{OPT}$ keeps $x$ in the cache until the next access to $x$ $\rightarrow x$ is resident

$a, b, c$ are residents

$$\sigma = a \ b \ c \ b \ a \ d \ c \ e \ f \ a \ c \ d \ c \ f \ a \ b \ a \ e$$

\[
\begin{array}{|c|c|c|}
\hline
a & b & c \\
\hline
\end{array}
\]
Assume $OPT$ brings a page $x$ to the cache at time $t$.

- Either $OPT$ evict $x$ before the next access to it $\rightarrow x$ is mortal.
- $OPT$ keeps $x$ in the cache until the next access to $x$ $\rightarrow x$ is resident

$a, b, c$ are residents

$$\sigma = a \ b \ c \ b \ a \ d \ c \ e \ f \ a \ c \ d \ c \ f \ a \ b \ a \ e$$
Assume $\text{OPT}$ brings a page $x$ to the cache at time $t$.

- Either $\text{OPT}$ evict $x$ before the next access to it $\rightarrow x$ is mortal.
- $\text{OPT}$ keeps $x$ in the cache until the next access to $x$ $\rightarrow x$ is resident

\[
\sigma = a \ b \ c \ b \ a \ d \ c \ e \ f \ a \ c \ d \ c \ f \ a \ b \ a \ e
\]

\[
\begin{array}{|c|c|c|}
\hline
a & b & c \\
\hline
\end{array}
\]
Assume $\text{OPT}$ brings a page $x$ to the cache at time $t$.

- Either $\text{OPT}$ evict $x$ before the next access to it $\rightarrow x$ is mortal.
- $\text{OPT}$ keeps $x$ in the cache until the next access to $x$ $\rightarrow x$ is resident

$a, c$ are residents $\quad b$ is mortal

$\sigma = a \quad b \quad c \quad b \quad a \quad d \quad c \quad e \quad f \quad a \quad c \quad d \quad c \quad f \quad a \quad b \quad a \quad e$

| a | b | c |
Assume $\text{Opt}$ brings a page $x$ to the cache at time $t$.

- Either $\text{Opt}$ evict $x$ before the next access to it $\rightarrow x$ is mortal.
- $\text{Opt}$ keeps $x$ in the cache until the next access to $x$ $\rightarrow x$ is resident

$$\sigma = a\ b\ c\ b\ a\ d\ c\ e\ f\ a\ c\ d\ c\ f\ a\ b\ a\ e$$

$a, c$ are residents  \hspace{1cm} $b$ is mortal
Assume $\text{OPT}$ brings a page $x$ to the cache at time $t$.

- Either $\text{OPT}$ evicts $x$ before the next access to it $\rightarrow x$ is mortal.
- $\text{OPT}$ keeps $x$ in the cache until the next access to $x$ $\rightarrow x$ is resident

\[ a, c \text{ are residents} \quad b \text{ is mortal} \]

\[ \sigma = a \ b \ c \ b \ a \ d \ c \ e \ f \ a \ c \ d \ c \ f \ a \ b \ a \ e \]
Assume $\text{OPT}$ brings a page $x$ to the cache at time $t$.

- Either $\text{OPT}$ evicts $x$ before the next access to it $\rightarrow x$ is mortal.
- $\text{OPT}$ keeps $x$ in the cache until the next access to $x \rightarrow x$ is resident.

$$\sigma = a \ b \ c \ b \ a \ d \ c \ e \ f \ a \ c \ d \ c \ f \ a \ b \ a \ e$$

```
\begin{array}{c|c|c}
\hline
  & a & d \\
\hline
\end{array}
```
Assume $\text{OPT}$ brings a page $x$ to the cache at time $t$.

- Either $\text{OPT}$ evict $x$ before the next access to it $\Rightarrow x$ is **mortal**.
- $\text{OPT}$ keeps $x$ in the cache until the next access to $x$ $\Rightarrow x$ is **resident**

$$\sigma = a \ b \ c \ b \ a \ d \ c \ e \ f \ a \ c \ d \ c \ f \ a \ b \ a \ e$$
If \( \text{OPT} \) has a hit for request \( x \) \( \Rightarrow \) \( x \) has been resident in cache since its last access to \( x \).

Consider an algorithm \( \text{ResMor} \) that evicts a mortal page if an eviction is required. \( \text{ResMor} \) always has the same resident pages as \( \text{OPT} \) in its cache. The mortal pages might be different. \( \text{OPT} \) and \( \text{ResMor} \) have the same cost. Assume \( \text{OPT} \) has smaller cost \( \Rightarrow \) there is a request to \( x \) that is a hit by \( \text{OPT} \) and a miss for \( \text{ResMor} \) \( \Rightarrow \) \( x \) is resident in \( \text{OPT} \) and not in \( \text{ResMor} \) \( \Rightarrow \) they maintain different resident pages (\( \text{ResMor} \) has evicted a resident page at some point) \( \Rightarrow \) contradiction.
Caching & advice (cntd.)

If $\text{OPT}$ has a hit for request $x \Rightarrow x$ has been resident in cache since its last access to $x$.

If $\text{OPT}$ has a fault for request $x \Rightarrow$ either it is the first access to $x$ or $x$ has been mortal after its previous access (so that it is evicted at some point).
If $\text{OPT}$ has a hit for request $x \Rightarrow x$ has been resident in cache since its last access to $x$.

If $\text{OPT}$ has a fault for request $x \Rightarrow$ either it is the first access to $x$ or $x$ has been mortal after its previous access (so that it is evicted at some point).

Consider an algorithm $\text{ResMor}$ that evicts a mortal page if an eviction is required.

- $\text{ResMor}$ always has the same resident pages as $\text{OPT}$ in its cache
- The mortal pages might be different.
Caching & advice (cntd.)

- If \( \text{OPT} \) has a hit for request \( x \) \( \Rightarrow \) \( x \) has been resident in cache since its last access to \( x \).

- If \( \text{OPT} \) has a fault for request \( x \) \( \Rightarrow \) either it is the first access to \( x \) or \( x \) has been mortal after its previous access (so that it is evicted at some point).

- Consider an algorithm \( \text{ResMor} \) that evicts a mortal page if an eviction is required.
  - \( \text{ResMor} \) always has the same resident pages as \( \text{OPT} \) in its cache
  - The mortal pages might be different.

- \( \text{OPT} \) and \( \text{ResMor} \) have the same cost
  - Assume \( \text{OPT} \) has smaller cost \( \Rightarrow \) there is a request to \( x \) that is a hit by \( \text{OPT} \) and a miss for \( \text{ResMor} \) \( \Rightarrow \) \( x \) is resident in \( \text{OPT} \) and not in \( \text{ResMore} \) \( \Rightarrow \) they maintain different resident pages (\( \text{ResMore} \) has evicted a resident page at some point) \( \Rightarrow \) contradiction
ResMor is an optimal algorithm that, instead of the whole sequence, only needs to know which pages are resident/mortal at each given time.
ResMor is an optimal algorithm that, instead of the whole sequence, only needs to know which pages are resident/mortal at each given time.

Assume with each request, there is one bit of advice that indicates whether the requested page is resident or mortal after the request.
ResMor is an optimal algorithm that, instead of the whole sequence, only needs to know which pages are resident/mortal at each given time.

Assume with each request, there is one bit of advice that indicates whether the requested page is resident or mortal after the request.

We can think of ResMor as an online algorithm with $n$ bits of advice.
What does the advice encode?

What is the size of advice?

How the algorithm works, provided by these bits of advice?

Why the algorithm has a competitive ratio of 1 (optimal here)?
What does the advice encode? For each request there is one bit of advice indicating whether the requested page is evicted before the next request to it (i.e., it is mortal) or not (i.e., it is resident).

What is the size of advice?

How the algorithm works, provided by these bits of advice?

Why the algorithm has a competitive ratio of 1 (optimal here)?
Caching Problem

Caching & advice

What does the advice encode? For each request there is one bit of advice indicating whether the requested page is evicted before the next request to it (i.e., it is mortal) or not (i.e., it is resident).

What is the size of advice? It is one bit for request, i.e., $n$ bits for a sequence of $n$ requests.

How the algorithm works, provided by these bits of advice?

Why the algorithm has a competitive ratio of 1 (optimal here)?
What does the advice encode? For each request there is one bit of advice indicating whether the requested page is evicted before the next request to it (i.e., it is mortal) or not (i.e., it is resident).

What is the size of advice? It is one bit for request, i.e., \( n \) bits for a sequence of \( n \) requests.

How the algorithm works, provided by these bits of advice? When eviction is required, it evicts any mortal page.

Why the algorithm has a competitive ratio of 1 (optimal here)?
Caching Problem

Caching & advice

- What does the advice encode? For each request there is one bit of advice indicating whether the requested page is evicted before the next request to it (i.e., it is mortal) or not (i.e., it is resident).

- What is the size of advice? It is one bit for request, i.e., $n$ bits for a sequence of $n$ requests.

- How the algorithm works, provided by these bits of advice? When eviction is required, it evicts any mortal page.

- Why the algorithm has a competitive ratio of 1 (optimal here)? It maintains the same resident pages as Opt; so in case of a hit by Opt there will be a hit by the algorithm.
Advice complexity of paging

- With $n$ bits of advice, one can achieve an optimal algorithm.
- With roughly $\log\left(\frac{r+1}{r+1}\right) \cdot n$ bits, one can achieve a competitive ratio of $r$
  - With roughly $0.27n$ bits, one can achieve a competitive ratio of 2.
  - With roughly $0.24n$ bits, one can achieve a competitive ratio of 3.
With $n$ bits of advice, one can achieve an optimal algorithm.

With roughly $\log\left(\frac{r+1}{r+1}\right) \cdot n$ bits, one can achieve a competitive ratio of $r$

- With roughly $0.27n$ bits, one can achieve a competitive ratio of 2.
- With roughly $0.24n$ bits, one can achieve a competitive ratio of 3.

For a potential project, do a survey on advice complexity of paging, and try to deduce new results!
$k$-Server Problem
**k-server problem**

- A **metric** is a set of points with a **distance** between each of pairs so that \( d(x, y) \leq d(x, z) + d(z, y) \).

  - E.g., a connected, undirected graph or a set of points in plane

\[
\sigma = < S, M, K, A, D, B, D, B, D >
\]

\[
\text{costs} = 2, 0, 2, 1, 1, 1, 1, 1, 1
\]
**Introduction**

**k-sever problem**

- A **metric** is a set of points with a **distance** between each of pairs so that \( d(x, y) \leq d(x, z) + d(z, y) \).
  - E.g., a connected, undirected graph or a set of points in plane

- We have a metric space of size \( m \)
  - \( k < m \) servers in the graph

\[ \sigma = < S \ M \ K \ A \ D \ B \ D \ B \ D > \]
\[ \text{costs} = 2 \ 0 \ 2 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \]
A metric is a set of points with a distance between each of pairs so that \( d(x, y) \leq d(x, z) + d(z, y) \).

E.g., a connected, undirected graph or a set of points in plane.

We have a metric space of size \( m \)

\( k < m \) servers in the graph

A sequence of \( n \) requests to the vertices of the graph

Each request should be served by a server.

Requests appear in an online manner.
**Introduction**

**$k$-server problem**

- **A metric** is a set of points with a **distance** between each of pairs so that $d(x, y) \leq d(x, z) + d(z, y)$.
  - E.g., a connected, undirected graph or a set of points in plane

- **We have a metric space of size $m$**
  - $k < m$ servers in the graph

- **A sequence of $n$ requests** to the vertices of the graph
  - Each request should be served by a server
  - Requests appear in an online manner

- **Minimize the total distance moved by servers**

\[ \sigma = \langle S, M, K, A, D, B, D, B, D \rangle \]
\[ \text{costs} = 2, 0, 2, 1, 1, 1, 1, 1, 1 \]
**$k$-sever problem**

- A **metric** is a set of points with a **distance** between each of pairs so that $d(x, y) \leq d(x, z) + d(z, y)$.
  - E.g., a connected, undirected graph or a set of points in plane
- We have a metric space of size $m$
  - $k < m$ servers in the graph
- A sequence of $n$ **requests** to the vertices of the graph
  - Each request should be served by a server
  - Requests appear in an online manner
- Minimize the total distance moved by servers

$$\sigma = < S \ M \ A \ D \ B \ D \ B \ D >$$
$$\text{costs} = 2 \ 0 \ 2 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1$$
Introduction

\textit{k-sever} problem

- A metric is a set of points with a distance between each of pairs so that \( d(x, y) \leq d(x, z) + d(z, y) \).
  - E.g., a connected, undirected graph or a set of points in plane

- We have a metric space of size \( m \)
  - \( k < m \) servers in the graph

- A sequence of \textit{n requests} to the vertices of the graph
  - Each request should be served by a server
  - Requests appear in an online manner

- Minimize the total distance moved by servers
**Introduction**

**k-sever problem**

- **A metric** is a set of points with a **distance** between each of pairs so that $d(x, y) \leq d(x, z) + d(z, y)$.
  - E.g., a connected, undirected graph or a set of points in plane

- We have a metric space of size $m$
  - $k < m$ servers in the graph

- A sequence of *n requests* to the vertices of the graph
  - Each request should be served by a server
  - Requests appear in an online manner

- Minimize the total distance moved by servers
Introduction

**k-sever problem**

- A **metric** is a set of points with a **distance** between each of pairs so that \( d(x, y) \leq d(x, z) + d(z, y) \).
  - E.g., a connected, undirected graph or a set of points in plane
- We have a metric space of size \( m \)
  - \( k < m \) servers in the graph
- A sequence of **requests** to the vertices of the graph
  - Each request should be served by a server
  - Requests appear in an online manner
- Minimize the total distance moved by servers
A metric is a set of points with a distance between each of pairs so that 
\( d(x, y) \leq d(x, z) + d(z, y) \).

- E.g., a connected, undirected graph or a set of points in plane

We have a metric space of size \( m \)

- \( k < m \) servers in the graph

A sequence of \( n \) requests to the vertices of the graph

- Each request should be served by a server
- Requests appear in an online manner

Minimize the total distance moved by servers
Introduction

**k-sever problem**

- A metric is a set of points with a distance between each of pairs so that \( d(x, y) \leq d(x, z) + d(z, y) \).
  - E.g., a connected, undirected graph or a set of points in plane
- We have a metric space of size \( m \)
  - \( k < m \) servers in the graph
- A sequence of \( n \) requests to the vertices of the graph
  - Each request should be served by a server
  - Requests appear in an online manner
- Minimize the total distance moved by servers

\[ \sigma = < S, M, K, A, D, B, D, B, D > \]
\[ \text{costs} = 2, 0, 2, 1, 1, 1, 1, 1, 1 \]
**$k$-sever problem**

- **A metric** is a set of points with a **distance** between each of pairs so that $d(x, y) \leq d(x, z) + d(z, y)$.
  - E.g., a connected, undirected graph or a set of points in plane

- We have a metric space of size $m$
  - $k < m$ servers in the graph

- A sequence of *n requests* to the vertices of the graph
  - Each request should be served by a server
  - Requests appear in an online manner

- Minimize the total distance moved by servers

\[ \sigma = \langle S, M, K, A, D, B, D, B, D \rangle \]
\[ \text{costs} = [2, 0, 2, 1, 1, 1, 1, 1, 1] \]
**k-sever problem**

- A **metric** is a set of points with a **distance** between each of pairs so that \( d(x, y) \leq d(x, z) + d(z, y) \).
  - E.g., a connected, undirected graph or a set of points in plane
- We have a metric space of size \( m \)
  - \( k < m \) servers in the graph
- A sequence of \( n \) requests to the vertices of the graph
  - Each request should be served by a server
  - Requests appear in an online manner
- Minimize the total distance moved by servers

\[
\sigma = < S \ M \ K \ A \ D \ B \ D \ B \ D > \\
costs = 2 \ 0 \ 2 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1
\]
Introduction

\textbf{\textit{k}}-server problem

- A \textbf{metric} is a set of points with a \textbf{distance} between each of pairs so that $d(x, y) \leq d(x, z) + d(z, y)$.
  - E.g., a connected, undirected graph or a set of points in plane
- We have a metric space of size $m$
  - $k < m$ servers in the graph
- A sequence of \textit{n requests} to the vertices of the graph
  - Each request should be served by a server
  - Requests appear in an online manner
- Minimize the total distance moved by servers

$\sigma = \langle S, M, K, A, D, B, D, B, D \rangle$
\text{costs} = 2 0 2 1 1 1 1 1 1
Introduction

The $k$-server problem

- What happens if we have a complete graph (a uniform metric)?
Introduction

The $k$-server problem

- What happens if we have a complete graph (a uniform metric)?
  - If there is a request to a vertex at which a server is located → there is no cost; otherwise, there is a cost of 1 to move a server to requested vertex.
The $k$-server problem

- What happens if we have a complete graph (a uniform metric)?
  - If there is a request to a vertex at which a server is located $\rightarrow$ there is no cost; otherwise, there is a cost of 1 to move a server to requested vertex.
  - Think of vertices as pages; vertices with servers on them are pages in the cache $\rightarrow$ caching problem.
Introduction

The $k$-server problem

- What happens if we have a complete graph (a uniform metric)?
  - If there is a request to a vertex at which a server is located → there is no cost; otherwise, there is a cost of 1 to move a server to requested vertex.
    - Think of vertices as pages; vertices with servers on them are pages in the cache → caching problem.

- Recall that for caching problem, we have:

  **Theorem**

  No deterministic algorithm can achieve a competitive ratio better than $k$, and LRU and FIFO achieve this ratio.

  No randomized algorithm can achieve a competitive ratio that is asymptotically better than $\Theta(\log k)$ and Mark algorithm achieves this.
Introduction

The $k$-server problem

- What happens if we have a complete graph (a uniform metric)?
  - If there is a request to a vertex at which a server is located → there is no cost; otherwise, there is a cost of 1 to move a server to requested vertex.
  - Think of vertices as pages; vertices with servers on them are pages in the cache → caching problem.

- Recall that for caching problem, we have:

  **Theorem**
  
  No deterministic algorithm can achieve a competitive ratio better than $k$, and LRU and FIFO achieve this ratio.
  
  No randomized algorithm can achieve a competitive ratio that is asymptotically better than $\Theta(\log k)$ and Mark algorithm achieves this.

- $k$-server problem has the right level of difficulty compared to paging (which is ‘too easy’) and Metrical Task Systems (another problem which is ‘too hard’)
Introduction

Greedy Algorithm

- Move the closest server to serve each request.

\[ \sigma = \langle S \ M \ K \ A \ D \ B \ D \ B \ D \rangle \]

\[ \text{costs} = \langle 2 \ 0 \ 2 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \rangle \]
Introduction

Greedy Algorithm

- Move the closest server to serve each request.

\[\sigma = \langle S, M, K, A, D, B, B, B, D \rangle\]

\[\text{costs} = 2, 0, 2, 1, 1, 1, 1, 1, 1\]
Greedy Algorithm

- Move the closest server to serve each request.

\[
\sigma = \langle S, M, K, A, D, B, D, B, D \rangle
\]

\[
\text{costs} = 2, 0, 2, 1, 1, 1, 1, 1, 1
\]
Greedy Algorithm

- Move the closest server to serve each request.

\[\sigma = \langle S, M, K, A, D, B, D, B, D \rangle\]

\[\text{costs} = 2 \quad 0 \quad 2 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \]
Greedy Algorithm

- Move the closest server to serve each request.

\[ \sigma = <S, M, K, A, D, B, D, B, D> \]
\[ \text{costs} = 2, 0, 2, 1, 1, 1, 1, 1, 1 \]
Introduction

Greedy Algorithm

- Move the closest server to serve each request.
- Is Greedy a good algorithm?
Greedy Algorithm

- Move the closest server to serve each request.
- Is Greedy a good algorithm?
  - what about the input \( \sigma = \langle B \ R \ B \ R \ldots \rangle \)?
    - For \( n \) requests, greedy incurs a cost of \( n \)
    - \( \text{OPT} \) moves another server from \( M \) to \( T \) at a cost of 3 and incurs no cost.
    - Competitive ratio will be at least \( \frac{n}{3} \) for this graph!
Greedy Algorithm

Theorem

For any graph of diameter $d$, the competitive ratio of greedy is at least $\frac{n}{2d}$.

- It holds for any graph, even a path!
- Consider two vertices $A$ and $B$ which are close to one server and further from other servers.
  - Greedy servers sequence $\langle A \ B \ A \ B \ \ldots \rangle$ by one server
Theorem

For any graph of diameter $d$, the competitive ratio of greedy is at least $\frac{n}{2d}$.

- It holds for any graph, even a path!
- Consider two vertices $A$ and $B$ which are close to one server and further from other servers.
  - Greedy servers sequence $\langle A \ B \ A \ B \ \ldots \rangle$ by one server
Greedy Algorithm

Theorem

For any graph of diameter \( d \), the competitive ratio of greedy is at least \( \frac{n}{2d} \).

- It holds for any graph, even a path!
- Consider two vertices \( A \) and \( B \) which are close to one server and further from other servers.
  - Greedy servers sequence \( \langle A \ B \ A \ B \ldots \rangle \) by one server
Greedy Algorithm

Theorem

For any graph of diameter $d$, the competitive ratio of greedy is at least $\frac{n}{2d}$.

- It holds for any graph, even a path!
- Consider two vertices $A$ and $B$ which are close to one server and further from other servers.
  - Greedy servers sequence $\langle A \ B \ A \ B \ \ldots \rangle$ by one server
Theorem

For any graph of diameter $d$, the competitive ratio of greedy is at least $\frac{n}{2d}$.

- It holds for any graph, even a path!
- Consider two vertices $A$ and $B$ which are close to one server and further from other servers.
  - Greedy servers sequence $\langle A\ B\ A\ B\ \ldots \rangle$ by one server

![Diagram showing vertices A, B, C, D, E, F, G with sequence 1: A, B, C, D, E, and 2: D, E, F, G]


**Introduction**

**Greedy Algorithm**

---

**Theorem**

For any graph of diameter $d$, the competitive ratio of greedy is at least $\frac{n}{2d}$.

- It holds for any graph, even a path!
- Consider two vertices $A$ and $B$ which are close to one server and further from other servers.
  - Greedy servers sequence $\langle A \ B \ A \ B \ \ldots \rangle$ by one server
  - The cost of Greedy is at least $n$
  - An optimal algorithm moves two servers to $A$ and $B$ at a cost of at most $2d$.
  - The competitive ratio of greedy becomes at least $\frac{n}{2d}$.
For general metrics

- No deterministic online algorithm can be better than $k$-competitive. (we see the proof in the next class)
$k$-server conjecture

**Conjecture**

**Conjecture:** for any metric space, there is a deterministic algorithm with competitive ratio of $k$.

- $k$-server conjecture is one of the big open problems in the context of online algorithms.
Conjecture

Conjecture: for any metric space, there is a deterministic algorithm with competitive ratio of k.

- k-server conjecture is one of the big open problems in the context of online algorithms.
  - Verified when $k = 2, m = k + 1, m = k + 2$, and trees.

- In the next class, we learn about potential function algorithm, which has a competitive ratio of $2k - 1$.