Announcements

- HW 1 due in class on Tue Jan 24
- Email me your homework partner name, or if you need a partner *today*
Greedy Algorithms

- Direct argument - MST
- Exchange argument - Caching
- Greedy approximation algorithms
Optimal Caching

Given a sequence of memory accesses, limited cache: How do you decide which cache element to evict?

Note: We are given future memory accesses for this problem, which is usually not the case. This is for an application of greedy algorithms
Optimal Caching: Example

Given a sequence of memory accesses, limited cache size, how do you decide which cache element to evict?

**Goal:** Minimize #main memory fetches
Optimal Caching: Example

**Goal:** Minimize #main memory fetches

Given a sequence of memory accesses, limited cache size, How do you decide which cache element to evict?

**Goal:** Minimize #main memory fetches
Optimal Caching: Example

Given a sequence of memory accesses, limited cache size, How do you decide which cache element to evict?

**Goal:** Minimize #main memory fetches
Optimal Caching: Example

Given a sequence of memory accesses, limited cache size, How do you decide which cache element to evict?

**Goal:** Minimize #main memory fetches
Optimal Caching: Example

Given a sequence of memory accesses, limited cache size, How do you decide which cache element to evict?

**Goal:** Minimize #main memory fetches
Optimal Caching: Example

Given a sequence of memory accesses, limited cache size, How do you decide which cache element to evict?

**Goal:** Minimize #main memory fetches
Optimal Caching: Example

Given a sequence of memory accesses, limited cache size, how do you decide which cache element to evict?

**Goal:** Minimize #main memory fetches
Optimal Caching

Farthest-First (FF) Schedule: Evict an item when needed. Evict the element which is accessed farthest down in the future

Theorem: The FF algorithm minimizes \#fetches
Optimal Caching

**Farthest-First (FF) Schedule:** Evict an item when needed. Evict the element which is accessed farthest down in the future

**Theorem:** The FF algorithm minimizes #fetches
Optimal Caching

**Farthest-First (FF) Schedule:** Evict an item when needed. Evict the element which is accessed farthest down in the future.

**Theorem:** The FF algorithm minimizes #fetches
Optimal Caching

Farthest-First (FF) Schedule: Evict an item when needed. Evict the element which is accessed farthest down in the future.

Theorem: The FF algorithm minimizes \#fetches.
Optimal Caching

**Farthest-First (FF) Schedule:** Evict an item when needed. Evict the element which is accessed farthest down in the future.

**Theorem:** The FF algorithm minimizes fetches.
Optimal Caching

Farthest-First (FF) Schedule: Evict an item when needed. Evict the element which is accessed farthest down in the future.

Theorem: The FF algorithm minimizes #fetches
Optimal Caching

**Farthest-First (FF) Schedule:** Evict an item when needed. Evict the element which is accessed farthest down in the future.

**Theorem:** The FF algorithm minimizes number of fetches.
**Optimal Caching**

<table>
<thead>
<tr>
<th>M</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>b</th>
<th>c</th>
<th>b</th>
<th>a</th>
<th>a</th>
</tr>
</thead>
</table>

**Memory Access Sequence**

<table>
<thead>
<tr>
<th>S₁</th>
<th>a</th>
<th>a</th>
<th>c</th>
<th>c</th>
<th>c</th>
<th>b</th>
<th>b</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>a</td>
</tr>
</tbody>
</table>

**Cache Contents**

<table>
<thead>
<tr>
<th>E</th>
<th></th>
<th></th>
<th>a</th>
<th></th>
<th></th>
<th></th>
<th>c</th>
<th></th>
</tr>
</thead>
</table>

**Evicted items**

**Farthest-First (FF) Schedule:** Evict an item when needed. Evict the element which is accessed farthest down in the future.

**Theorem:** The FF algorithm minimizes fetches.
**Caching: Reduced Schedule**

<table>
<thead>
<tr>
<th>M</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>b</th>
<th>c</th>
<th>b</th>
<th>a</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>a</td>
<td>a</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>b</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>E</td>
<td>-</td>
<td>-</td>
<td>a</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>c</td>
<td>-</td>
</tr>
</tbody>
</table>

An eviction schedule is **reduced** if it fetches an item $x$ only when it is accessed.

**Fact:** For any $S$, there is a reduced schedule $S^*$ which makes at most as many fetches as $S$.
Caching: Reduced Schedule

An eviction schedule is **reduced** if it fetches an item x only when it is accessed.

**Fact:** For any S, there is a reduced schedule $S^*$ with at most as many fetches as S.

To convert S to $S^*$: Be lazy!
Optimal Caching Theorem

**Theorem:** Suppose a reduced schedule $S_j$ makes the same decisions as SFF from $t=1$ to $t=j$. Then, there exists a reduced schedule $S_{j+1}$ s.t:
1. $S_{j+1}$ makes **same decision** as SFF from $t=1$ to $t=j+1$
2. $\#\text{fetches}(S_{j+1}) \leq \#\text{fetches}(S_j)$

Suppose you claim a magic schedule schedule $S_M$ makes less fetches than SFF Then, we can construct a sequence of schedules:
$S_M = S_0, S_1, S_2, \ldots, S_n = SFF$ such that:
(1) $S_j$ agrees with SFF from $t=1$ to $t = j$
(2) $\#\text{fetches}(S_{j+1}) \leq \#\text{fetches}(S_j)$

What does this say about $\#\text{fetches}(SFF)$ relative to $\#\text{fetches}(S_M)$?
**Theorem:** Suppose a reduced schedule $S_j$ makes the same decisions as SFF from $t=1$ to $t=j$. Then, there exists a reduced schedule $S_{j+1}$ s.t:
1. $S_{j+1}$ makes **same decision** as SFF from $t=1$ to $t=j+1$
2. $\#\text{fetches}(S_{j+1}) \leq \#\text{fetches}(S_j)$

- $S_j$
- $S_{j+1}$
- SFF

![Diagram showing the comparison of schedules and cache](cache-diagram.png)

Cache $t=j$
Theorem: Suppose a reduced schedule $S_j$ makes the same decisions as SFF from $t=1$ to $t=j$. Then, there exists a reduced schedule $S_{j+1}$ s.t:
1. $S_{j+1}$ makes \textbf{same decision} as SFF from $t=1$ to $t=j+1$
2. $\#\text{fetches}(S_{j+1}) \leq \#\text{fetches}(S_j)$

### Case 1:
No cache miss at $t=j+1$. $S_{j+1} = S_j$
Theorem: Suppose a reduced schedule \( S_j \) makes the same decisions as SFF from \( t=1 \) to \( t=j \). Then, there exists a reduced schedule \( S_{j+1} \) s.t:
1. \( S_{j+1} \) makes **same decision** as SFF from \( t=1 \) to \( t=j+1 \)
2. \( \#fetches(S_{j+1}) \leq \#fetches(S_j) \)

**Case 2:** Cache miss at \( t=j+1 \), \( S_j \) and SFF evict same item. \( S_{j+1} = S_j \)
Caching: FF Schedules

**Theorem:** Suppose a reduced schedule $S_j$ makes the same decisions as SFF from $t=1$ to $t=j$. Then, there exists a reduced schedule $S_{j+1}$ s.t:

1. $S_{j+1}$ makes **same decision** as SFF from $t=1$ to $t=j+1$
2. $\#\text{fetches}(S_{j+1}) \leq \#\text{fetches}(S_j)$

**Case 3a:** Cache miss at $t=j+1$. $S_j$ evicts a, SFF evicts b. $S_{j+1}$ also evicts b. Next there is a request to d, and $S_j$ evicts b. Make $S_{j+1}$ evict a, bring in d.
Theorem: Suppose a reduced schedule \( S_j \) makes the same decisions as SFF from \( t=1 \) to \( t=j \). Then, there exists a reduced schedule \( S_{j+1} \) s.t:
1. \( S_{j+1} \) makes **same decision** as SFF from \( t=1 \) to \( t=j+1 \)
2. \( \#\text{fetches}(S_{j+1}) \leq \#\text{fetches}(S_j) \)

**Case 3b:** Cache miss at \( t=j+1 \). \( S_j \) evicts a, SFF evicts b. \( S_{j+1} \) also evicts b.

Next there is a request to a, and \( S_j \) evicts b. \( S_{j+1} \) does nothing.
Caching: FF Schedules

**Theorem:** Suppose a reduced schedule $S_j$ makes the same decisions as SFF from $t=1$ to $t=j$. Then, there exists a reduced schedule $S_{j+1}$ s.t:
1. $S_{j+1}$ makes **same decision** as SFF from $t=1$ to $t=j+1$
2. $\#\text{fetches}(S_{j+1}) \leq \#\text{fetches}(S_j)$

**Case 3c:** Cache miss at $t=j+1$. $S_j$ evicts a, SFF evicts b. $S_{j+1}$ also evicts b. Next there is a request to a, and $S_j$ evicts d. $S_{j+1}$ evicts d and brings in b. Now convert $S_{j+1}$ to the reduced version of this schedule.
Caching: FF Schedules

**Theorem:** Suppose a reduced schedule $S_j$ makes the same decisions as SFF from $t=1$ to $t=j$. Then, there exists a reduced schedule $S_{j+1}$ s.t:

1. $S_{j+1}$ makes **same decision** as SFF from $t=1$ to $t=j+1$
2. $\#\text{fetches}(S_{j+1}) \leq \#\text{fetches}(S_j)$

**Case 3d:** Cache miss at $t=j+1$. $S_j$ evicts a, SFF evicts b. $S_{j+1}$ also evicts b. 
Next there is a request to b. **Cannot happen** as a is accessed before b!
Theorem: Suppose a reduced schedule $S_j$ makes the same decisions as SFF from $t=1$ to $t=j$. Then, there exists a reduced schedule $S_{j+1}$ s.t:
1. $S_{j+1}$ makes same decision as SFF from $t=1$ to $t=j+1$
2. $\#\text{fetches}(S_{j+1}) \leq \#\text{fetches}(S_j)$

Case 1: No cache miss at $t=j+1$. $S_{j+1} = S_j$

Case 2: Cache miss at $t=j+1$, $S_j$ and SFF evict same item. $S_{j+1} = S_j$

Case 3a: Cache miss at $t=j+1$. $S_j$ evicts a, SFF evicts b. $S_{j+1}$ also evicts b. Next there is a request to d, and $S_j$ evicts b. Make $S_{j+1}$ evict a, bring in d.

Case 3b: Cache miss at $t=j+1$. $S_j$ evicts a, SFF evicts b. $S_{j+1}$ also evicts b. Next there is a request to a, and $S_j$ evicts b. $S_{j+1}$ does nothing.

Case 3c: Cache miss at $t=j+1$. $S_j$ evicts a, SFF evicts b. $S_{j+1}$ also evicts b. Next there is a request to a, and $S_j$ evicts d. $S_{j+1}$ evicts d and brings in b. Now convert $S_{j+1}$ to the reduced version of this schedule.

Case 3d: Cache miss at $t=j+1$. $S_j$ evicts a, SFF evicts b. $S_{j+1}$ also evicts b. Next there is a request to b. Cannot happen as a is accessed before b!
**Theorem:** Suppose a reduced schedule $S_j$ makes the same decisions as SFF from $t=1$ to $t=j$. Then, there exists a reduced schedule $S_{j+1}$ s.t:

1. $S_{j+1}$ makes **same decision** as SFF from $t=1$ to $t=j+1$
2. $\#\text{fetches}(S_{j+1}) \leq \#\text{fetches}(S_j)$

Suppose you claim a magic schedule schedule $S_M$ makes less fetches than SFF. Then, we can construct a sequence of schedules:

$S_M = S_0, S_1, S_2, ..., S_n = SFF$ such that:

(1) $S_j$ agrees with SFF from $t=1$ to $t = j$
(2) $\#\text{fetches}(S_{j+1}) \leq \#\text{fetches}(S_j)$

What does this say about $\#\text{fetches}(SFF)$ relative to $\#\text{fetches}(S_M)$?
Greedy Algorithms

- Direct argument - MST
- Exchange argument - Caching
- Greedy approximation algorithms
Greedy Approximation Algorithms

- k-Center
- Set Cover
Approximation Algorithms

• Optimization problems, eg, MST, Shortest paths

• What do we optimize?

• What if we do not have enough resources to compute the optimal solution?
Approximation Algorithms

For an instance $I$ of a **minimization problem**, let:

$$A(I) = \text{value of solution by algorithm } A$$

$$OPT(I) = \text{value of optimal solution}$$

Approximation ratio($A$) = $\max_I A(I)/OPT(I)$

$A$ is an **approx. algorithm** if approx-ratio($A$) is bounded
Approximation Algorithms

For an instance \( I \) of a **minimization problem**, let:

\[
A(I) = \text{value of solution by algorithm } A
\]

\[
OPT(I) = \text{value of optimal solution}
\]

Approximation ratio(\( A \)) = \( \max_I A(I)/OPT(I) \)

\( A \) is an **approx. algorithm** if approx-ratio(\( A \)) is bounded

**Higher** approximation ratio means **worse** algorithm
Greedy Approximation Algorithms

- k-Center
- Set Cover
k-Center Problem

Given \textbf{n towns} on a map
Find how to place \textbf{k shopping malls} such that:
Drive to the nearest mall from any town is shortest
**k-Center Problem**

Given **n towns** on a map
Find how to place **k shopping malls** such that:
Drive to the nearest mall from any town is shortest

$k=3$
k-Center Problem

Given **n points** in a **metric space**
Find **k centers** such that distance between any point and its closest center is as small as possible

**Metric Space:**
Point set w/ distance fn \(d\)

**Properties of \(d\):**
- \(d(x, y) \geq 0\)
- \(d(x, y) = d(y, x)\)
- \(d(x, y) \leq d(x, z) + d(y, z)\)

**NP Hard** in general
A Greedy Algorithm: Farthest-first traversal

1. Pick $C = \{x\}$, for an arbitrary point $x$
2. Repeat until $C$ has $k$ centers:
   - Let $y$ maximize $d(y, C)$, where $d(y, C) = \min_{x \in C} d(x, y)$
   - $C = C \cup \{y\}$

$k=3$
A Greedy Algorithm: Farthest-first traversal

1. Pick $C = \{x\}$, for an arbitrary point $x$
2. Repeat until $C$ has $k$ centers:
   
   Let $y$ maximize $d(y, C)$, where
   
   $d(y, C) = \min_{x \in C} d(x, y)$

   $C = C \cup \{y\}$

$k=3$
A Greedy Algorithm: Farthest-first traversal

1. Pick $C = \{x\}$, for an arbitrary point $x$
2. Repeat until $C$ has $k$ centers:
   - Let $y$ maximize $d(y, C)$, where
     \[ d(y, C) = \min_{x \in C} d(x, y) \]
   - $C = C \cup \{y\}$
A Greedy Algorithm: Farthest-first traversal

1. Pick $C = \{x\}$, for an arbitrary point $x$
2. Repeat until $C$ has $k$ centers:
   - Let $y$ maximize $d(y, C)$, where $d(y, C) = \min_{x \in C} d(x, y)$
   - $C = C \cup \{y\}$
A Greedy Algorithm: Farthest-first traversal

1. Pick $C = \{x\}$, for an arbitrary point $x$
2. Repeat until $C$ has $k$ centers:
   Let $y$ maximize $d(y, C)$, where
   $$d(y, C) = \min_{x \in C} d(x, y)$$
   $$C = C \cup \{y\}$$

$k=3$
Farthest-first Traversal

Is farthest-first traversal always optimal?

Theorem: Approx. ratio of farthest-first traversal is 2
Facts on Analyzing Approx. Algorithms

- Need to reason about the algorithm relative to the optimal solution
- Example: Optimal solution has value greater than or equal to some value $X$
Farthest-first (FF) Traversal

**Theorem:** Approx. ratio of FF-traversal is 2
Define, for any instance: \( r = \max_x d(x, C) \)
\[ q = \arg\max_x d(x, C) \]

**Metric Space:**
Point set w/ **distance fn** \( d \)

**Properties of \( d \):**
- \( d(x, y) \geq 0 \)
- \( d(x, y) = d(y, x) \)
- \( d(x, y) \leq d(x, z) + d(y, z) \)

For a set \( S \),
\[ d(x, S) = \min_y \in S d(x, y) \]

**FF-traversal:**
Pick \( C = \{x\} \), arbitrary \( x \)
Repeat until \( C \) has \( k \) centers:
Let \( y \) **maximize** \( d(y, C) \)
\[ C = C \cup \{y\} \]

**Property 1.** Solution value of FF-traversal = \( r \)
Farthest-first (FF) Traversal

**Metric Space:**
Point set w/ distance fn $d$

**Properties of $d$:**
\begin{itemize}
  \item $d(x, y) \geq 0$
  \item $d(x, y) = d(y, x)$
  \item $d(x, y) \leq d(x, z) + d(y, z)$
\end{itemize}

For a set $S$, $d(x, S) = \min_{y \in S} d(x, y)$

**FF-traversal:**
Pick $C = \{x\}$, arbitrary $x$
Repeat until $C$ has $k$ centers:
  Let $y$ maximize $d(y, C)$
  $C = C \cup \{y\}$

**Theorem:** Approx. ratio of FF-traversal is 2
Define, for any instance: $r = \max_x d(x, C)$
$q = \arg\max_x d(x, C)$

**Property 1.** Solution value of FF-traversal = $r$

**Property 2.** There are at least $k+1$ points $S$ s.t. each pair has distance $\geq r$
**Theorem:** Approx. ratio of FF-traversal is 2
Define, for any instance:  \( r = \max_x d(x, C) \)
\( q = \arg\max_x d(x, C) \)

**Property 1.** Solution value of FF-traversal = \( r \)

**Property 2.** There are at least \( k+1 \) points \( S \) s.t each pair has distance \( \geq r \), where \( S = C \cup \{q\} \).
Farthest-first (FF) Traversal

Theorem: Approx. ratio of FF-traversal is 2
Define, for any instance: \( r = \max_x d(x, C) \)
\( q = \arg\max_x d(x, C) \)

Metric Space:
Point set w/ distance fn \( d \)

Properties of \( d \):
• \( d(x, y) \geq 0 \)
• \( d(x, y) = d(y, x) \)
• \( d(x, y) \leq d(x, z) + d(y, z) \)

For a set \( S \),
\( d(x, S) = \min_{y \in S} d(x, y) \)

FF-traversal:
Pick \( C = \{x\} \), arbitrary \( x \)
Repeat until \( C \) has \( k \) centers:
Let \( y \) \textbf{maximize} \( d(y, C) \)
\( C = C \cup \{y\} \)

Property 1. Solution value of FF-traversal = \( r \)
Property 2. There are at least \( k+1 \) points \( S \) s.t each pair has distance \( \geq r \), where \( S = C \cup \{q\} \).
Property 3. The Optimal solution must assign at least two points \( x, y \) in \( S \) to the same center \( c \)

What is \( \max(d(x, c), d(y, c)) \)?
Farthest-first (FF) Traversal

Metric Space:
Point set w/ distance fn \( d \)

Properties of \( d \):
- \( d(x, y) \geq 0 \)
- \( d(x, y) = d(y, x) \)
- \( d(x, y) \leq d(x, z) + d(y, z) \)

For a set \( S \),
\[
    d(x, S) = \min_{y \in S} d(x, y)
\]

FF-traversal:
Pick \( C = \{x\} \), arbitrary \( x \)
Repeat until \( C \) has \( k \) centers:
- Let \( y \) maximize \( d(y, C) \)
- \( C = C \cup \{y\} \)

Theorem: Approx. ratio of FF-traversal is 2
Define, for any instance:
\[
    r = \max_x d(x, C) \\
    q = \arg\max_x d(x, C)
\]

Property 3. The optimal solution must assign at least two points \( x, y \) in \( S \) to the same center \( c \)
What is \( \max(d(x, c), d(y, c)) \)?

From property of \( d \),
\[
    d(x, c) + d(y, c) \geq d(x, y) \\
    \max(d(x, c), d(y, c)) \geq d(x, y)/2
\]
Farthest-first (FF) Traversal

**Metric Space:**
Point set with distance function \(d\)

**Properties of \(d\):**
- \(d(x, y) \geq 0\)
- \(d(x, y) = d(y, x)\)
- \(d(x, y) \leq d(x, z) + d(y, z)\)

For a set \(S\),
- \(d(x, S) = \min_{y \in S} d(x, y)\)

**FF-traversal:**
Pick \(C = \{x\}\), arbitrary \(x\)
Repeat until \(C\) has \(k\) centers:
- Let \(y\) maximize \(d(y, C)\)
- \(C = C \cup \{y\}\)

**Theorem:** Approx. ratio of FF-traversal is 2
Define, for any instance: \(r = \max_x d(x, C)\)
\(q = \arg\max_x d(x, C)\)

**Property 1.** Solution value of FF-traversal = \(r\)

**Property 2.** There are at least \(k+1\) points \(S\) s.t.
each pair has distance \(\geq r\), where \(S = C \cup \{q\}\)

**Property 3.** The optimal solution must assign at least two points \(x, y\) in \(S\) to the same center \(c\)
- \(\max(d(x, c), d(y, c)) \geq d(x, y)/2 \geq r/2\)

**Property 4.** Thus, Opt. solution has value \(\geq r/2\)
Summary: k center

Given **n points** in a **metric space**
Find **k centers** such that distance between any point and its closest center is as small as possible

**FF-Traversal Algorithm:**
1. Pick $C = \{x\}$, for an arbitrary point $x$
2. Repeat until $C$ has $k$ centers:
   - Let $y$ maximize $d(y, C)$, where
     $$d(y, C) = \min_{x \in C} d(x, y)$$
   - $C = C \cup \{y\}$

$k$-center is **NP hard**, but **approx. ratio** of FF-traversal is $2$
Applications of k-center:

- Facility-location problems
- Clustering
Greedy Approximation Algorithms

- k-Center
- Set Cover
Set Cover Problem

Given:
- Universe $U$ with $n$ elements
- Collection $C$ of sets of elements of $U$

Find the smallest subset $C^*$ of $C$ that covers all of $U$

*NP Hard* in general
Set Cover Problem

Given:
- Universe \( U \) with \( n \) elements
- Collection \( C \) of sets of elements of \( U \)

Find the smallest subset \( C^* \) of \( C \) that covers all of \( U \)

NP Hard in general
A Greedy Set-Cover Algorithm

\[
C^* = \{ \} \\
\text{Repeat until all of } U \text{ is covered:} \\
\quad \text{Pick the set } S \text{ in } C \text{ with highest # of uncovered elements} \\
\quad \text{Add } S \text{ to } C^*
\]
A Greedy Set-Cover Algorithm

\[ C^* = \{ \} \]
Repeat until all of U is covered:
   Pick the set S in C with highest \# of uncovered elements
   Add S to C*
A Greedy Set-Cover Algorithm

\[ C^* = \{ \} \]
Repeat until all of \( U \) is covered:
   - Pick the set \( S \) in \( C \) with highest \# of uncovered elements
   - Add \( S \) to \( C^* \)
A Greedy Set-Cover Algorithm

C* = {}  
Repeat until all of U is covered:  
   Pick the set S in C with highest # of uncovered elements  
   Add S to C*
A Greedy Set-Cover Algorithm

C* = {}  
Repeat until all of U is covered:
   Pick the set S in C with highest # of uncovered elements  
   Add S to C*
A Greedy Set-Cover Algorithm

C* = { }
Repeat until all of U is covered:
   Pick the set S in C with highest # of uncovered elements
   Add S to C*
A Greedy Set-Cover Algorithm

\[ C^* = \{ \} \]
Repeat until all of U is covered:
   Pick the set S in C with highest # of uncovered elements
   Add S to C*

\[ C^* = \{ \} \]
A Greedy Set-Cover Algorithm

\[ C^* = \{ \} \]
Repeat until all of \( U \) is covered:
  - Pick the set \( S \) in \( C \) with highest \# of uncovered elements
  - Add \( S \) to \( C^* \)

Greedy: \#sets=7
A Greedy Set-Cover Algorithm

$C^* = \{ \}$

Repeat until all of $U$ is covered:

- Pick the set $S$ in $C$ with highest # of uncovered elements
- Add $S$ to $C^*$

**Greedy:** #sets=7

**OPT:** #sets=5
**Greedy Set-Cover Algorithm**

**Theorem:** If optimal set cover has k sets, then greedy selects $\leq k \ln n$ sets

**Greedy Algorithm:**

- $C^* = \{ \}$
- Repeat until $U$ is covered:
  - Pick $S$ in $C$ with highest # of uncovered elements
  - Add $S$ to $C^*$

Define:
- $n(t) = \# \text{uncovered elements after step } t \text{ in greedy}$

**Property 1:** There is some $S$ that covers at least $n(t)/k$ of the uncovered elements

**Property 2:** $n(t+1) \leq n(t)(1 - 1/k)$

**Property 3:** $n(T) \leq n(1 - 1/k)^T < 1$, when $T = k \ln n$
Summary: set cover

Given: Universe U with \(n\) elements
Collection C of sets of elements of U
Find the smallest subset \(C^*\) of C that covers all of U

Greedy Algorithm:
\[ C^* = \{ \} \]
Repeat until U is covered:
   Pick S in C with highest \# of uncovered elements

Set-cover is \textbf{NP hard}, but approx. ratio of Greedy is \(O(\log n)\)
The Maximum Coverage Problem

Given:
- Universe U with n elements
- Collection C of sets of elements of U

Find a subset C* of C of size k that covers as many elements of U as possible

A different version of Set-cover

NP hard

Greedy algorithm also has a good approx-ratio
Applications of Set Cover and Max. Coverage

• Facility location problems
• Submodular optimization
Greedy Algorithms

• Direct argument - MST
• Exchange argument - Caching
• Greedy approximation algorithms
  • k-center, set-cover