CSE 202: Design and Analysis of Algorithms

Lecture 4

Instructor: Kamalika Chaudhuri
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:

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

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

Approximation ratio(A) = \( \max_I A(I)/\text{OPT}(I) \)

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-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
   
   $d(y, C) = \min_{x \in C} d(x, y)$
   
   $C = C \cup \{y\}$
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
     $d(y, C) = \min_{x \in C} d(x, y)$
   - $C = C \cup \{y\}$

$k=3$
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
   
   \[ d(y, C) = \min_{x \in C} d(x, y) \]
   
   \( C = C \cup \{y\} \)
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 optimal solution
• Need to reason about the approx. algorithm relative to the optimal solution
**Farthest-first (FF) Traversal**

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

**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 = \( r \)

**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 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 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 \( d \),
\[
d(x, c) + d(y, c) \geq d(x, y)
\]
\[
\max(d(x, c), d(y, c)) \geq \frac{d(x, y)}{2}
\]
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 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 \(\geq r/2\)
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
     \[
     d(y, C) = \min_{x \in C} d(x, y)
     \]
   - \( C = C U \{y\} \)

**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

\[ C^* = \{ \} \]

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 \( C^* \)
A Greedy Set-Cover Algorithm

\[ C^* = \emptyset \]

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*

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:

$n(t) =$ #uncovered elements after step $t$ 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 **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
Algorithm Design Paradigms

• Exhaustive Search

• **Greedy Algorithms**: Build a solution incrementally piece by piece

• **Divide and Conquer**: Divide into parts, solve each part, combine results

• **Dynamic Programming**: Divide into subtasks, perform subtask by size. Combine smaller subtasks to larger ones

• **Hill-climbing**: Start with a solution, improve it
A Simple Divide and Conquer Example: Mergesort

Recurrence: \( T(n) = 2T(n/2) + cn \)

Solution: \( T(n) = O(n \log n) \)
Divide and Conquer

• Integer Multiplication

• Strassen’s Matrix Multiplication

• Closest pair of points on a Plane
How to multiply two $n$-bit numbers?

1. Create array of $n$ intermediate sums
2. Add up the sums

Time per addition = $O(n)$
Total time = $O(n^2)$.

Can we do better?
Problem: How to multiply two n-bit numbers $x$ and $y$?

$x = x_L 2^{n/2} + x_R$

$y = y_L 2^{n/2} + y_R$
A Simple Divide and Conquer

**Problem:** How to multiply two n-bit numbers $x$ and $y$?

$x = x_L 2^{n/2} + x_R$

$y = y_L 2^{n/2} + y_R$

$xy = (x_L 2^{n/2} + x_R)(y_L 2^{n/2} + y_R)$

$= x_L y_L 2^n + (x_R y_L + x_L y_R) 2^{n/2} + x_R y_R$
Problem: How to multiply two n-bit numbers x and y?

\[ x = x_L 2^{n/2} + x_R \]

\[ y = y_L 2^{n/2} + y_R \]

\[ xy = (x_L 2^{n/2} + x_R)(y_L 2^{n/2} + y_R) = x_L y_L 2^n + (x_R y_L + x_L y_R) 2^{n/2} + x_R y_R \]
A Simple Divide and Conquer

**Problem:** How to multiply two n-bit numbers $x$ and $y$?

$x = x_L 2^{n/2} + x_R$

$y = y_L 2^{n/2} + y_R$

$xy = (x_L 2^{n/2} + x_R)(y_L 2^{n/2} + y_R)$

$= x_L y_L 2^n + (x_R y_L + x_L y_R) 2^{n/2} + x_R y_R$

**Operations:**
1. 4 multiplications of n/2 bit #s
2. Shifting by n bits
3. 3 additions

**Recurrence:**
$T(n) = 4T(n/2) + O(n)$
A Simple Divide and Conquer

**Problem:** How to multiply two n-bit numbers $x$ and $y$?

$x = x_L 2^{n/2} + x_R$

$y = y_L 2^{n/2} + y_R$

$xy = (x_L 2^{n/2} + x_R)(y_L 2^{n/2} + y_R)$

$= x_L y_L 2^n + (x_R y_L + x_L y_R) 2^{n/2} + x_R y_R$

What is the base case?

**Operations:**

1. 4 multiplications of n/2 bit #s
2. Shifting by n bits
3. 3 additions

**Recurrence:**

$T(n) = 4T(n/2) + O(n)$

$T(n) = O(n^2)$
**Problem:** How to multiply two n-bit numbers $x$ and $y$?

\[
x = x_L \ 2^{n/2} + x_R
\]

\[
y = y_L \ 2^{n/2} + y_R
\]

\[
xy = (x_L \ 2^{n/2} + x_R)(y_L \ 2^{n/2} + y_R)
\]

\[
= x_L y_L \ 2^n + (x_R \ y_L + x_L y_R) \ 2^{n/2} + x_R y_R
\]

**Need:** $x_L y_L$, $x_R y_R$, $(x_R y_L + x_L y_R)$

**Computed by 3 multiplications as:**

\[
(x_R y_L + x_L y_R) = (x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R
\]
A Better Divide and Conquer

Problem: How to multiply two n-bit numbers $x$ and $y$?

$x = x_L 2^{n/2} + x_R$

$y = y_L 2^{n/2} + y_R$

$xy = (x_L 2^{n/2} + x_R)(y_L 2^{n/2} + y_R)$

$= x_L y_L 2^n + (x_R y_L + x_L y_R) 2^{n/2} + x_R y_R$

Need: $x_L y_L, x_R y_R, (x_R y_L + x_L y_R)$

Computed by 3 multiplications as:

$(x_R y_L + x_L y_R) = (x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R$

Operations:

1. 3 multiplications of $n/2$ bit #s
2. Shifting by $n$ bits
3. 6 additions
**A Better Divide and Conquer**

**Problem:** How to multiply two n-bit numbers $x$ and $y$?

$x = x_L 2^{n/2} + x_R$

$y = y_L 2^{n/2} + y_R$

$xy = (x_L 2^{n/2} + x_R)(y_L 2^{n/2} + y_R)$

$= x_L y_L 2^n + (x_R y_L + x_L y_R) 2^{n/2} + x_R y_R$

Need: $x_L y_L, x_R y_R, (x_R y_L + x_L y_R)$

Computed by 3 multiplications as:

$(x_R y_L + x_L y_R) = (x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R$

**Operations:**
1. 3 multiplications of $n/2$ bit #s
2. Shifting by $n$ bits
3. 6 additions

**Recurrence:**

$T(n) = 3T(n/2) + O(n)$
**A Better Divide and Conquer**

**Problem:** How to multiply two \(n\)-bit numbers \(x\) and \(y\)?

\[
x = x_L 2^{n/2} + x_R
\]
\[
y = y_L 2^{n/2} + y_R
\]

\[
xy = (x_L 2^{n/2} + x_R)(y_L 2^{n/2} + y_R)
\]
\[
= x_L y_L 2^n + (x_R y_L + x_L y_R) 2^{n/2} + x_R y_R
\]

Need: \(x_L y_L, x_R y_R, (x_R y_L + x_L y_R)\)

Computed by 3 multiplications as:
\[
(x_R y_L + x_L y_R) = (x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R
\]

**Operations:**
1. 3 multiplications of \(n/2\) bit \#s
2. Shifting by \(n\) bits
3. 6 additions

**Recurrence:**
\[
T(n) = 3T(n/2) + O(n)
\]
\[
T(n) = O(n^{1.59})
\]
A Better Divide and Conquer

**Problem:** How to multiply two n-bit numbers \( x \) and \( y \)?

\[
x = x_L \cdot 2^{n/2} + x_R \\
y = y_L \cdot 2^{n/2} + y_R
\]

\[
xy = (x_L \cdot 2^{n/2} + x_R)(y_L \cdot 2^{n/2} + y_R) \\
= x_Ly_L \cdot 2^n + (x_Ry_L + x_ly_R) \cdot 2^{n/2} + x_Ry_R
\]

Need: \( x_Ly_L, x_Ry_R, (x_Ry_L + x_ly_R) \)

Computed by 3 multiplications as:

\[
(x_Ry_L + x_ly_R) = (x_L + x_R)(y_L + y_R) - x_Ly_L - x_Ry_R
\]

Best: \( O(n \log n \cdot 2^{O(\log^* n)}) \) [Furer07]

**Operations:**
1. 3 multiplications of n/2 bit #s
2. Shifting by n bits
3. 6 additions

**Recurrence:**
\[
T(n) = 3T(n/2) + O(n) \\
T(n) = O(n^{1.59})
\]
Divide and Conquer

- Integer Multiplication
- Strassen’s Matrix Multiplication
- Closest pair of points on a Plane
**Problem:** Given two $n \times n$ matrices $X$ and $Y$, compute $Z = XY$

$$Z_{ij} = \sum X_{ik} Y_{kj}$$

**Naive Method:** $O(n^3)$ time
A Simple Divide and Conquer

**Problem:** Given two $n \times n$ matrices $X$ and $Y$, compute $Z = XY$

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix}, \quad Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix}$$

$$Z = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$$

A..H: $n/2 \times n/2$

**Algorithm 1:**
1. Compute $AE, BG, CE, DG, AF, BH, CF, DH$
2. Compute $AE + BG, CE + DG, AF + BH, CF + DH$

**Operations:** 8 $n/2 \times n/2$ matrix multiplications, 4 additions

**Recurrence:** $T(n) = 8T(n/2) + O(n^2)$ \quad $T(n) = O(n^3)$
Strassen’s Algorithm

Problem: Given two $n \times n$ matrices $X$ and $Y$, compute $Z = XY$

\[
X = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \\
Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix}
\]

\[
Z = \begin{bmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{bmatrix}
\]

$A..H: n/2 \times n/2$

\[
P_1 = A(F - H) \\
P_2 = (A + B)H \\
P_3 = (C + D)E \\
P_4 = D(G - E) \\
P_5 = (A + D)(E + H) \\
P_6 = (B - D)(G + H) \\
P_7 = (A - C)(E + F)
\]

Operations: 7 $n/2 \times n/2$ matrix multiplications, $O(1)$ additions

Recurrence: $T(n) = 7 T(n/2) + O(n^2) \\
T(n) = O(n^{2.81})$
Strassen’s Algorithm

Problem: Given two $n \times n$ matrices $X$ and $Y$, compute $Z = XY$

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \quad Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix}$$

$$Z = \begin{bmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{bmatrix}$$

A..H: n/2 x n/2

$$P_1 = A(F - H) \quad P_3 = (C + D)E \quad P_5 = (A + D)(E + H)$$
$$P_2 = (A + B)H \quad P_4 = D(G - E) \quad P_6 = (B - D)(G + H)$$

Operations: 7 $n/2 \times n/2$ matrix multiplications, $O(1)$ additions

Recurrence: $T(n) = 7T(n/2) + O(n^2) \quad T(n) = O(n^{2.81})$

Best: $O(n^{2.37})$ by Coppersmith-Winograd[90]
Divide and Conquer

• Integer Multiplication
• Strassen’s Matrix Multiplication
• Closest pair of points on a Plane
Closest Pair of Points on a Plane

Given a set $P$ of $n$ points on the plane, find the two points $p$ and $q$ in $P$ s.t $d(p, q)$ is minimum.
Closest Pair of Points on a Plane

Given a set $P$ of $n$ points on the plane, find the two points $p$ and $q$ in $P$ s.t $d(p, q)$ is minimum

Naive solution $= O(n^2)$
What about one dimension?

Given a set $P$ of $n$ points on the line, find the two points $p$ and $q$ in $P$ s.t $d(p, q)$ is minimum.

\[ \bullet \bullet \bullet \bullet \bullet \bullet \bullet \bullet \bullet \bullet \bullet \bullet \bullet \]

$P$ \hspace{1cm} $q$
What about one dimension?

Given a set P of n points on the line, find the two points p and q in P such that \(d(p, q)\) is minimum.

Property: The closest points are adjacent in sorted order.
What about one dimension?

Given a set $P$ of $n$ points on the line, find the two points $p$ and $q$ in $P$ s.t $d(p, q)$ is minimum.

Property: The closest points are adjacent in sorted order.

Algorithm 1
1. Sort the points
2. Find a point $p_i$ in the sorted set s.t $d(p_i, p_{i+1})$ is minimum
What about one dimension?

Given a set $P$ of $n$ points on the line, find the two points $p$ and $q$ in $P$ s.t $d(p, q)$ is minimum

Property: The closest points are adjacent in sorted order

Algorithm 1
1. Sort the points
2. Find a point $p_i$ in the sorted set s.t $d(p_i, p_{i+1})$ is minimum

Running Time = $O(n \log n)$
Does this work in 2D?

(a, b) : closest in x-coordinate
(a, q) : closest in y-coordinate
(p, q) : closest

Sorting the points by x or y coordinate, and looking at adjacent pairs does not work!