CSCI 5451 Fall 2015

Week 10 Notes

Professor Ellen Gethner

October 18, 2015

Network flow problem, continued from last week

- Reminder of the Network Flow Question:

Network flow problem, continued from last week

- Reminder of the Network Flow Question:

- **The Network Flow Problem.** Given a flow network G with capacity constraint function c, find the flow of maximum value.

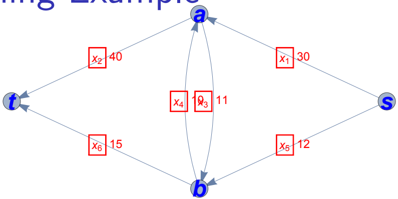
Network flow problem, continued from last week

- Reminder of the Network Flow Question:
- **The Network Flow Problem.** Given a flow network G with capacity constraint function c, find the flow of maximum value.
- For a long time, there was no efficient solution to solve the above problem, but there was (and is) a technique called Linear Programming that was used to find a solution.

Network flow problem, continued from last week

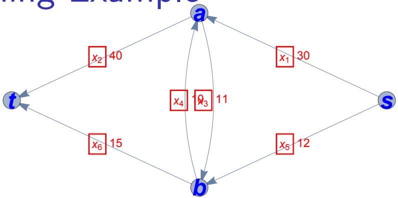
- Reminder of the Network Flow Question:
- **The Network Flow Problem.** Given a flow network G with capacity constraint function c, find the flow of maximum value.
- For a long time, there was no efficient solution to solve the above problem, but there was (and is) a technique called Linear Programming that was used to find a solution.
- Let's see how linear programming works on network flows in an example.

Linear Programming Example



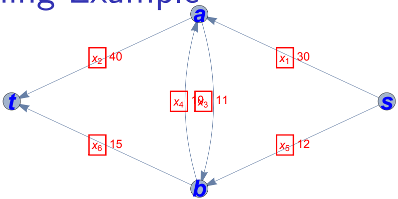
- The number in the □ box is the flow and can change.

Linear Programming Example



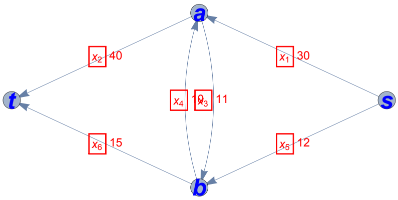
- The number in the $\square$ box is the flow and can change.
- The number in the non-box is the capacity and can't change.

Linear Programming Example



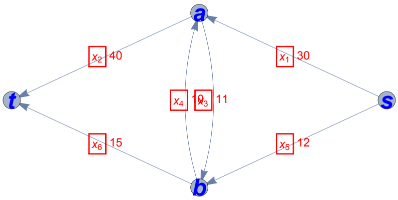
- The number in the $\square$ box is the flow and can change.
- The number in the non-box is the capacity and can't change.
- **Problem.** Maximize $x_1 + x_5$ subject to
  1. $0 \leq x_1 \leq 30$
  2. $0 \leq x_2 \leq 40$
  3. $0 \leq x_3 \leq 11$
  4. $0 \leq x_4 \leq 10$
  5. $0 \leq x_5 \leq 12$
  6. $0 \leq x_6 \leq 15$ together with
  7. $x_2 + x_3 - x_1 - x_4 = 0$, and
  8. $x_6 + x_4 - x_5 - x_3 = 0$.

Linear Programming example, continued



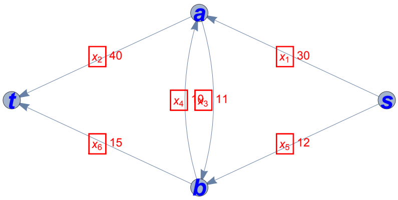
- *Mathematica* has a built-in linear programming command, but the above example can be done on scratch paper!

Linear Programming example, continued



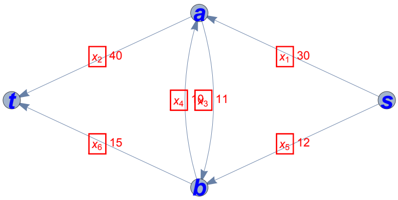
- *Mathematica* has a built-in linear programming command, but the above example can be done on scratch paper!
- **Solution.** Pause

Linear Programming example, continued



- *Mathematica* has a built-in linear programming command, but the above example can be done on scratch paper!
- **Solution.** Pause
- $x_1 + x_2 = 42$ by way of $x_1 = 30$, $x_2 = 40$, $x_3 = 0$, $x_4 = 10$, $x_5 = 12$, and $x_6 = 2$.

Linear Programming example, continued



- *Mathematica* has a built-in linear programming command, but the above example can be done on scratch paper!

- **Solution.** Pause

- $x_1 + x_2 = 42$ by way of $x_1 = 30$, $x_2 = 40$, $x_3 = 0$, $x_4 = 10$, $x_5 = 12$, and $x_6 = 2$.

- Why do we know that we've maximized the flow?

Thoughts about linear programming

- In most cases linear programming is inefficient, but often we have no other techniques to rely upon and so are grateful for an opportunity for a solution.

Thoughts about linear programming

- In most cases linear programming is inefficient, but often we have no other techniques to rely upon and so are grateful for an opportunity for a solution.

- The **Network Flow Problem** happens to be an optimization problem that *can* be solved using linear programming, but there is a much better technique.

Thoughts about linear programming

- In most cases linear programming is inefficient, but often we have no other techniques to rely upon and so are grateful for an opportunity for a solution.

- The **Network Flow Problem** happens to be an optimization problem that *can* be solved using linear programming, but there is a much better technique.

- **History.** Before 1956, it was not known whether or not for a given arbitrary (weighted, directed) graph G that a solution exists.

Thoughts about linear programming

- In most cases linear programming is inefficient, but often we have no other techniques to rely upon and so are grateful for an opportunity for a solution.
- The **Network Flow Problem** happens to be an optimization problem that *can* be solved using linear programming, but there is a much better technique.
- **History.** Before 1956, it was not known whether or not for a given arbitrary (weighted, directed) graph G that a solution exists.
- In 1956, the first algorithm, a graph algorithm was proposed by Ford and Fulkerson, and over the years, their solution was improved.

Network Flow Hall of Fame (see Herb Wilf's book, ch. 3)

Author(s)	Year	Complexity
Ford, Fulkerson	1956	not known
Edmonds, Karp	1969	$O(E^2 V)$
Dinic	1970	$O(EV^2)$
Karzonov	1973	$O(V^3)$
Cherkassky	1976	$O(V^2\sqrt{E})$
Malhotra, et al	1978	$O(V^3)$
Galil	1978	$O(V^{\frac{5}{3}} E^{\frac{2}{3}})$
Galil, Naamed	1979	$O(EV log^2(V))$
Skeater, Tarjan	1980	$O(EV log(V))$
Goldberg, Tarjan	1985	$O(EV log(\frac{V^2}{E}))$
King, Rao, Tarjan	1994	$O(EV log(\frac{E}{V}) log(V^V))$
Goldberg, Rao	1998	$O(E min(V^{\frac{2}{3}}, \sqrt{E}) log(\frac{V^2}{E}) log(U)$[1]
Orlin, King, Rao, Tarjan	2013	$O(VE)$ with extra constraints[2]

[1] U is the maximum capacity of the network

[2] Orlin, James B. (2013). "Max flows in O(nm) time, or better". STOC '13 Proceedings of the forty-fifth annual ACM symposium on Theory of computing: 765-774.

Connection Between the Ford/Fulkerson Algorithm and a Famous Theorem

- The proof of correctness of the Ford/Fulkerson[3] algorithm to solve the network flow problem is related to the **Max Flow/Min Cut Theorem**.

[3] see Wilf's book chapter 3 for the full algorithm

Connection Between the Ford/Fulkerson Algorithm and a Famous Theorem

- The proof of correctness of the Ford/Fulkerson[3] algorithm to solve the network flow problem is related to the **Max Flow/Min Cut Theorem**.
- Before stating the theorem, we need one more piece of machinery, a *cut*, which is a set of edges that separates source s from sink t:

[3] see Wilf's book chapter 3 for the full algorithm

Connection Between the Ford/Fulkerson Algorithm and a Famous Theorem

- The proof of correctness of the Ford/Fulkerson[3] algorithm to solve the network flow problem is related to the **Max Flow/Min Cut Theorem**.

- Before stating the theorem, we need one more piece of machinery, a *cut*, which is a set of edges that separates source s from sink t:

- **Definition (cut).**

[3] see Wilf's book chapter 3 for the full algorithm

Connection Between the Ford/Fulkerson Algorithm and a Famous Theorem

- The proof of correctness of the Ford/Fulkerson[3] algorithm to solve the network flow problem is related to the **Max Flow/Min Cut Theorem**.
- Before stating the theorem, we need one more piece of machinery, a *cut*, which is a set of edges that separates source s from sink t:
- **Definition (cut).**
  - Let $A \subset V(G)$ such that $s \in A$ and $t \notin A$ and

[3] see Wilf's book chapter 3 for the full algorithm

Connection Between the Ford/Fulkerson Algorithm and a Famous Theorem

- The proof of correctness of the Ford/Fulkerson[3] algorithm to solve the network flow problem is related to the **Max Flow/Min Cut Theorem**.

- Before stating the theorem, we need one more piece of machinery, a *cut*, which is a set of edges that separates source s from sink t:

- **Definition (cut).**
  - Let $A \subset V(G)$ such that $s \in A$ and $t \notin A$ and
  - let $\overline{A} = A \setminus V(G)$ (ie, $\overline{A}$ is the complement of A)

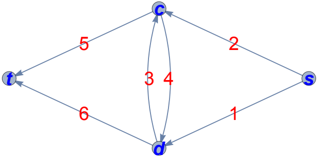
[3] see Wilf's book chapter 3 for the full algorithm

Connection Between the Ford/Fulkerson Algorithm and a Famous Theorem

- The proof of correctness of the Ford/Fulkerson[3] algorithm to solve the network flow problem is related to the **Max Flow/Min Cut Theorem**.
- Before stating the theorem, we need one more piece of machinery, a *cut*, which is a set of edges that separates source s from sink t:
- **Definition (cut).**
  - Let $A \subset V(G)$ such that $s \in A$ and $t \notin A$ and
  - let $\overline{A} = A \setminus V(G)$ (ie, $\overline{A}$ is the complement of A)
  - Then a **cut** is the set of edges $\{vw \in E(G) : v \in A,\ w \in \overline{A}\}$

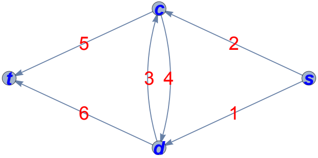
[3] see Wilf's book chapter 3 for the full algorithm

Example of a weighted graph G and all possible cuts



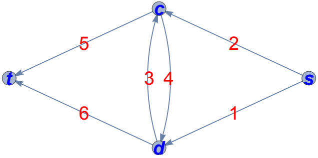
1. $A = \{s\}$, $\overline{A} = \{c, d, t\} \Rightarrow$ the corresponding cut is $\{sc, sd\}$.

Example of a weighted graph G and all possible cuts



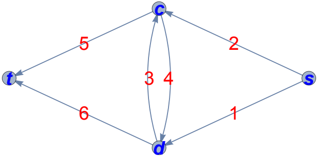
1. $A = \{s\}$, $\overline{A} = \{c, d, t\} \Rightarrow$ the corresponding cut is $\{sc, sd\}$.
2. $A = \{s, c\}$, $\overline{A} = \{d, t\} \Rightarrow$ the corresponding cut is $\{sd, cd, ct\}$.

Example of a weighted graph G and all possible cuts



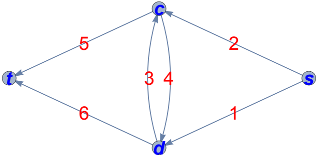
1. $A = \{s\}$, $\overline{A} = \{c, d, t\} \Rightarrow$ the corresponding cut is $\{sc, sd\}$.
2. $A = \{s, c\}$, $\overline{A} = \{d, t\} \Rightarrow$ the corresponding cut is $\{sd, cd, ct\}$.
3. $A = \{s, d\}$, $\overline{A} = \{c, t\} \Rightarrow$ the corresponding cut is $\{sc, dc, dt\}$.

Example of a weighted graph G and all possible cuts



1. $A = \{s\}$, $\overline{A} = \{c, d, t\} \Rightarrow$ the corresponding cut is $\{sc, sd\}$.
2. $A = \{s, c\}$, $\overline{A} = \{d, t\} \Rightarrow$ the corresponding cut is $\{sd, cd, ct\}$.
3. $A = \{s, d\}$, $\overline{A} = \{c, t\} \Rightarrow$ the corresponding cut is $\{sc, dc, dt\}$.
4. $A = \{s, c, d\}$, $\overline{A} = \{t\} \Rightarrow$ the corresponding cut is $\{ct, dt\}$.

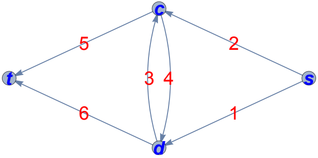
Example of a weighted graph G and all possible cuts



1. $A = \{s\}$, $\overline{A} = \{c, d, t\} \Rightarrow$ the corresponding cut is $\{sc, sd\}$.
2. $A = \{s, c\}$, $\overline{A} = \{d, t\} \Rightarrow$ the corresponding cut is $\{sd, cd, ct\}$.
3. $A = \{s, d\}$, $\overline{A} = \{c, t\} \Rightarrow$ the corresponding cut is $\{sc, dc, dt\}$.
4. $A = \{s, c, d\}$, $\overline{A} = \{t\} \Rightarrow$ the corresponding cut is $\{ct, dt\}$.

- **Definition.** The **capacity** of a cut is the sum of the capacities of its edges.

Example of a weighted graph G and all possible cuts



1. $A = \{s\}$, $\overline{A} = \{c, d, t\} \Rightarrow$ the corresponding cut is $\{sc, sd\}$.
2. $A = \{s, c\}$, $\overline{A} = \{d, t\} \Rightarrow$ the corresponding cut is $\{sd, cd, ct\}$.
3. $A = \{s, d\}$, $\overline{A} = \{c, t\} \Rightarrow$ the corresponding cut is $\{sc, dc, dt\}$.
4. $A = \{s, c, d\}$, $\overline{A} = \{t\} \Rightarrow$ the corresponding cut is $\{ct, dt\}$.

- **Definition.** The **capacity** of a cut is the sum of the capacities of its edges.
- Thus, for example, the capacity of **cut 1** is 3, and the capacity of **cut 4** is 11.

Max Flow/ Min Cut Theorem

- The correctness of the Ford-Fulkerson algorithm depends on the following theorem.

Max Flow/ Min Cut Theorem

- The correctness of the Ford-Fulkerson algorithm depends on the following theorem.
- **Theorem.** The value of a the maximum flow in a flow network G is the minimum capacity over all cuts in G.

New Topic: Keeping Secrets

- The art of cryptography is the art of sending and receiving messages while making sure that

New Topic: Keeping Secrets

- The art of cryptography is the art of sending and receiving messages while making sure that
- an evesdropper cannot read or compromise the message, and

New Topic: Keeping Secrets

- The art of cryptography is the art of sending and receiving messages while making sure that
- an evesdropper cannot read or compromise the message, and
- the recipient receives the correct message.

New Topic: Keeping Secrets

- The art of cryptography is the art of sending and receiving messages while making sure that
- an evesdropper cannot read or compromise the message, and
- the recipient receives the correct message.
- The Greek word cryptos means secret or hidden.

Keeping Secrets

- The process has two components:

Keeping Secrets

- The process has two components:

  1. Disguise the original plaintext message by way of an encryption or cipher; the disguised message is called a cryptogram.

Keeping Secrets

- The process has two components:

  1. Disguise the original plaintext message by way of an encryption or cipher; the disguised message is called a cryptogram.
  2. And the legitimate recipient must know some procedure to translate the cryptogram back to the original plaintext message. This process is called decryption and the recipient must have the encryption key.

Cryptanalysis

- The field of cryptography has a sister called cryptanalysis, which is the art of breaking a cipher;

Cryptanalysis

- The field of cryptography has a sister called cryptanalysis, which is the art of breaking a cipher;
- this is any method used to decrypt a cryptogram without holding the decryption key.

Cryptanalysis

- The field of cryptography has a sister called cryptanalysis, which is the art of breaking a cipher;
- this is any method used to decrypt a cryptogram without holding the decryption key.
- For example, linguistics and the use of frequency analysis may be enough to crack an encryption such as the Ceasar Cipher.

Cryptanalysis

- The field of cryptography has a sister called cryptanalysis, which is the art of breaking a cipher;
- this is any method used to decrypt a cryptogram without holding the decryption key.
- For example, linguistics and the use of frequency analysis may be enough to crack an encryption such as the Ceasar Cipher.
- In a Ceasar Cipher, one simply shifts the entire alphabet by a given fixed integer to scramble the message.

Cryptanalysis

- The field of cryptography has a sister called cryptanalysis, which is the art of breaking a cipher;
- this is any method used to decrypt a cryptogram without holding the decryption key.
- For example, linguistics and the use of frequency analysis may be enough to crack an encryption such as the Ceasar Cipher.
- In a Ceasar Cipher, one simply shifts the entire alphabet by a given fixed integer to scramble the message.
- Then since, for example, in the English language, the most frequently used symbol is the letter e, in an encrypted message, one simply looks for the most frequently used letter and from that, one can then determine the value of the shift.

Pubic Key Cryptography: Current Applications

- An interesting source, among others, is `http://www.laits.utexas.edu/~anorman/BUS.FOR/course.mat/SSim/life.html`,

Pubic Key Cryptography: Current Applications

- An interesting source, among others, is `http://www.laits.utexas.edu/~anorman/BUS.FOR/course.mat/SSim/life.html`,
- where brief descriptions of applications such as authentication, e-commerce, hard-disk encryption, are given.

Pubic Key Cryptography: Current Applications

- An interesting source, among others, is `http://www.laits.utexas.edu/~anorman/BUS.FOR/course.mat/SSim/life.html`,
- where brief descriptions of applications such as authentication, e-commerce, hard-disk encryption, are given.
- **Exercise.** Surf the web for more applications of cryptography.

Primality Testing

- Before moving on to a public key cryptographic algorithm, let's take a tangential side trip and think about

Primality Testing

- Before moving on to a public key cryptographic algorithm, let's take a tangential side trip and think about
- primality testing, which is a key component in most cryptographic tools.

A Side-Trip: Algorithms for Primality Testing

- **Algorithm Trial(n)**
- **Input.** $n \in \mathbb{Z}$; $n \geq 2$
- **Output.**

$$\begin{cases} 1 & \text{if } n \text{ is composite} \\ 0 & \text{if } n \text{ is prime} \end{cases}$$

- $i = 2$
- **while** $i^2 \leq n$ **repeat** (a *divisor of* n *must be* $\leq \sqrt{n}$)
- **if** $i | n$
- **then return** 1
- $i = i + 1$
- **return** 0

A Side-Trip: Algorithms for Primality Testing

- **Algorithm Trial(n)**
- **Input.** $n \in \mathbb{Z}$; $n \geq 2$
- **Output.**

$$\begin{cases} 1 & \text{if } n \text{ is composite} \\ 0 & \text{if } n \text{ is prime} \end{cases}$$

- $i = 2$
- **while** $i^2 \leq n$ **repeat** (*a divisor of n must be* $\leq \sqrt{n}$)
- **if** $i | n$
- **then return** 1
- $i = i + 1$
- **return** 0

- **Analysis.** Is the algorithm correct? If so, what is the runtime?

Algorithm Trial, analysis

- **Exercise.** Prove that Algorithm Trial is correct (not much to do for this).

Algorithm Trial, analysis

- **Exercise.** Prove that Algorithm Trial is correct (not much to do for this).

- **Example input.** Suppose n is a decimal number that is 62 digits long.

Algorithm Trial, analysis

- **Exercise.** Prove that Algorithm Trial is correct (not much to do for this).

- **Example input.** Suppose n is a decimal number that is 62 digits long.

- Then $\sqrt{n}$ is about 32 digits long (or $O(10^{32})$).

Algorithm Trial, analysis

- **Exercise.** Prove that Algorithm Trial is correct (not much to do for this).
- **Example input.** Suppose n is a decimal number that is 62 digits long.
- Then $\sqrt{n}$ is about 32 digits long (or $O(10^{32})$).
- In that case, the loop runs for about 10^{32} rounds.

Algorithm Trial, analysis

- **Exercise.** Prove that Algorithm Trial is correct (not much to do for this).
- **Example input.** Suppose n is a decimal number that is 62 digits long.
- Then $\sqrt{n}$ is about 32 digits long (or $O(10^{32})$).
- In that case, the loop runs for about 10^{32} rounds.
- Even after the naive speedups (if $2 \not| n$ remove multiples of 2, if $3 \not| n$ remove multiples of 3, etc)

Algorithm Trial, analysis

- **Exercise.** Prove that Algorithm Trial is correct (not much to do for this).
- **Example input.** Suppose n is a decimal number that is 62 digits long.
- Then $\sqrt{n}$ is about 32 digits long (or $O(10^{32})$).
- In that case, the loop runs for about 10^{32} rounds.
- Even after the naive speedups (if $2 \not| n$ remove multiples of 2, if $3 \not| n$ remove multiples of 3, etc)
- and assuming that a computer can carry out a trial division in one nanosecond ($=$ 1 billionth of a second)

Algorithm Trial, analysis

- **Exercise.** Prove that Algorithm Trial is correct (not much to do for this).
- **Example input.** Suppose n is a decimal number that is 62 digits long.
- Then $\sqrt{n}$ is about 32 digits long (or $O(10^{32})$).
- In that case, the loop runs for about 10^{32} rounds.
- Even after the naive speedups (if $2 \not| n$ remove multiples of 2, if $3 \not| n$ remove multiples of 3, etc)
- and assuming that a computer can carry out a trial division in one nanosecond ($=$ 1 billionth of a second)
- then how long would the computation take in this example?

Testing a 62 digit integer for primality: how long?

- **Computations.** One day $= 24$ hours $= 24 \times 60 = 1440$ minutes $= 86400$ seconds.

Testing a 62 digit integer for primality: how long?

- **Computations.** One day $= 24$ hours $= 24 \times 60 = 1440$ minutes $= 86400$ seconds.

- Each second is 10^9 nanoseconds, and thus one day is $86400 \times 10^9 \cong 10^{13}$ nanoseconds.

Testing a 62 digit integer for primality: how long?

- **Computations.** One day $= 24$ hours $= 24 \times 60 = 1440$ minutes $= 86400$ seconds.

- Each second is 10^9 nanoseconds, and thus one day is $86400 \times 10^9 \cong 10^{13}$ nanoseconds.

- In that case, since we have 10^{32} operations to perform, the length of time of the whole computation would take

Testing a 62 digit integer for primality: how long?

- **Computations.** One day $= 24$ hours $= 24 \times 60 = 1440$ minutes $= 86400$ seconds.
- Each second is 10^9 nanoseconds, and thus one day is $86400 \times 10^9 \cong 10^{13}$ nanoseconds.
- In that case, since we have 10^{32} operations to perform, the length of time of the whole computation would take
- $\frac{10^{32} \text{ operations}}{10^{13} \text{operations per day}} \cong 10^{19}$ days $\cong O(10^{16})$ years.

Testing a 62 digit integer for primality: how long?

- **Computations.** One day $= 24$ hours $= 24 \times 60 = 1440$ minutes $= 86400$ seconds.
- Each second is 10^9 nanoseconds, and thus one day is $86400 \times 10^9 \cong 10^{13}$ nanoseconds.
- In that case, since we have 10^{32} operations to perform, the length of time of the whole computation would take
- $\frac{10^{32} \text{ operations}}{10^{13} \text{operations per day}} \cong 10^{19}$ days $\cong O(10^{16})$ years.
- Yikes!

Testing a 62 digit integer for primality: how long?

- **Computations.** One day $= 24$ hours $= 24 \times 60 = 1440$ minutes $= 86400$ seconds.
- Each second is 10^9 nanoseconds, and thus one day is $86400 \times 10^9 \cong 10^{13}$ nanoseconds.
- In that case, since we have 10^{32} operations to perform, the length of time of the whole computation would take
- $\frac{10^{32} \text{ operations}}{10^{13} \text{operations per day}} \cong 10^{19}$ days $\cong O(10^{16})$ years.
- Yikes!
- **Exercise.** Generalize the argument above to determine a big Oh runtime for Algorithm Trial for an arbitrary positive integer n.

Euler's Theorem

- An important tool in two upcoming algorithms is Euler's Theorem.

Euler's Theorem

- An important tool in two upcoming algorithms is Euler's Theorem.
- Yes, he's the same Euler from planar graph theory fame, but the new theorem is from the field of Number Theory.

Euler's Theorem

- An important tool in two upcoming algorithms is Euler's Theorem.
- Yes, he's the same Euler from planar graph theory fame, but the new theorem is from the field of Number Theory.
- **Definition.** Let $\phi(n)$ denote the number of integers b such that $1 \leq b \leq n$ such that $gcd(b, n) = 1$.

Euler's Theorem

- An important tool in two upcoming algorithms is Euler's Theorem.
- Yes, he's the same Euler from planar graph theory fame, but the new theorem is from the field of Number Theory.
- **Definition.** Let $\phi(n)$ denote the number of integers b such that $1 \leq b \leq n$ such that $gcd(b, n) = 1$.
- The function ϕ is called Euler's Phi Function.

Euler's Theorem

- An important tool in two upcoming algorithms is Euler's Theorem.
- Yes, he's the same Euler from planar graph theory fame, but the new theorem is from the field of Number Theory.
- **Definition.** Let $\phi(n)$ denote the number of integers b such that $1 \leq b \leq n$ such that $gcd(b, n) = 1$.
- The function ϕ is called Euler's Phi Function.
- Examples at white board.

Euler's Theorem

- **Euler's Theorem.** Let $n \in \mathbb{N}$ and suppose $a \in \mathbb{Z}$ with $gcd(a, n) = 1$. Then $a^{\phi(n)} \equiv 1 \pmod{n}$.

Euler's Theorem

- **Euler's Theorem.** Let $n \in \mathbb{N}$ and suppose $a \in \mathbb{Z}$ with $gcd(a, n) = 1$. Then $a^{\phi(n)} \equiv 1 \pmod{n}$.

- **Proof.** Let $\{a_1, a_2, \ldots, a_{\phi(n)}\}$ be the set of integers in the range from 1 to n that are relatively prime to n.

Euler's Theorem

- **Euler's Theorem.** Let $n \in \mathbb{N}$ and suppose $a \in \mathbb{Z}$ with $gcd(a, n) = 1$. Then $a^{\phi(n)} \equiv 1 \pmod{n}$.

- **Proof.** Let $\{a_1, a_2, \ldots, a_{\phi(n)}\}$ be the set of integers in the range from 1 to n that are relatively prime to n.

- Consider the set $\{aa_1, aa_2, \ldots, aa_{\phi(n)}\}$.

Euler's Theorem

- **Euler's Theorem.** Let $n \in \mathbb{N}$ and suppose $a \in \mathbb{Z}$ with $gcd(a, n) = 1$. Then $a^{\phi(n)} \equiv 1 \pmod{n}$.

- **Proof.** Let $\{a_1, a_2, \dots, a_{\phi(n)}\}$ be the set of integers in the range from 1 to n that are relatively prime to n.

- Consider the set $\{aa_1, aa_2, \dots, aa_{\phi(n)}\}$.

- Since $gcd(a_i, n) = 1 \ \forall i = 1, 2, \dots \phi(n)$ and $gcd(a, n) = 1$, we have $gcd(aa_i, n) = 1$, as well.

Euler's Theorem

- **Euler's Theorem.** Let $n \in \mathbb{N}$ and suppose $a \in \mathbb{Z}$ with $gcd(a, n) = 1$. Then $a^{\phi(n)} \equiv 1 \pmod{n}$.

- **Proof.** Let $\{a_1, a_2, \ldots, a_{\phi(n)}\}$ be the set of integers in the range from 1 to n that are relatively prime to n.

- Consider the set $\{aa_1, aa_2, \ldots, aa_{\phi(n)}\}$.

- Since $gcd(a_i, n) = 1 \ \forall i = 1, 2, \ldots \phi(n)$ and $gcd(a, n) = 1$, we have $gcd(aa_i, n) = 1$, as well.

- **Claim.** The two sets $L_1 = \{a_1, a_2, \ldots, a_{\phi(n)}\}$ and $L_2 = \{aa_1, aa_2, \ldots, aa_{\phi(n)}\} \pmod{n}$ are the same.

Euler's Theorem

- **Euler's Theorem.** Let $n \in \mathbb{N}$ and suppose $a \in \mathbb{Z}$ with $gcd(a, n) = 1$. Then $a^{\phi(n)} \equiv 1 \pmod{n}$.
- **Proof.** Let $\{a_1, a_2, \ldots, a_{\phi(n)}\}$ be the set of integers in the range from 1 to n that are relatively prime to n.
- Consider the set $\{aa_1, aa_2, \ldots, aa_{\phi(n)}\}$.
- Since $gcd(a_i, n) = 1 \; \forall i = 1, 2, \ldots \phi(n)$ and $gcd(a, n) = 1$, we have $gcd(aa_i, n) = 1$, as well.
- **Claim.** The two sets $L_1 = \{a_1, a_2, \ldots, a_{\phi(n)}\}$ and $L_2 = \{aa_1, aa_2, \ldots, aa_{\phi(n)}\} \pmod{n}$ are the same.
- **Proof of claim.** It suffices to show that the set L_2 has no duplicate elements. Why?

Proof of Euler's Theorem, continued

- BWOC suppose $\exists i, j$ with $i \neq j$ and $1 \leq i, j \leq \phi(n)$ such that $aa_i \equiv aa_j \pmod{n}$.

Proof of Euler's Theorem, continued

- BWOC suppose $\exists i, j$ with $i \neq j$ and $1 \leq i, j \leq \phi(n)$ such that $aa_i \equiv aa_j \pmod{n}$.
- Then $aa_i - aa_j \equiv 0 \pmod{n}$.

Proof of Euler's Theorem, continued

- BWOC suppose $\exists i, j$ with $i \neq j$ and $1 \leq i, j \leq \phi(n)$ such that $aa_i \equiv aa_j \pmod{n}$.
- Then $aa_i - aa_j \equiv 0 \pmod{n}$.
- In that case $a(a_i - a_j) \equiv 0 \pmod{n}$ and therefore $n | (a(a_i - a_j)$

Proof of Euler's Theorem, continued

- BWOC suppose $\exists i, j$ with $i \neq j$ and $1 \leq i, j \leq \phi(n)$ such that $aa_i \equiv aa_j \pmod{n}$.
- Then $aa_i - aa_j \equiv 0 \pmod{n}$.
- In that case $a(a_i - a_j) \equiv 0 \pmod{n}$ and therefore $n | (a(a_i - a_j)$
- $\Rightarrow a_i = a_j$ because $gcd(a, n) = 1$ and $1 \leq a_i, a_j \leq n - 1$.

Proof of Euler's Theorem, continued

- BWOC suppose $\exists i, j$ with $i \neq j$ and $1 \leq i, j \leq \phi(n)$ such that $aa_i \equiv aa_j \pmod{n}$.
- Then $aa_i - aa_j \equiv 0 \pmod{n}$.
- In that case $a(a_i - a_j) \equiv 0 \pmod{n}$ and therefore $n | (a(a_i - a_j)$
- $\Rightarrow a_i = a_j$ because $gcd(a, n) = 1$ and $1 \leq a_i, a_j \leq n - 1$.
- This completes the proof of the claim.

Proof of Euler's Theorem, continued

- BWOC suppose $\exists i, j$ with $i \neq j$ and $1 \leq i, j \leq \phi(n)$ such that $aa_i \equiv aa_j \pmod{n}$.
- Then $aa_i - aa_j \equiv 0 \pmod{n}$.
- In that case $a(a_i - a_j) \equiv 0 \pmod{n}$ and therefore $n | (a(a_i - a_j)$
- $\Rightarrow a_i = a_j$ because $gcd(a, n) = 1$ and $1 \leq a_i, a_j \leq n - 1$.
- This completes the proof of the claim.
- Now on to the remainder of the proof of Euler's Theorem

Proof of Euler's Theorem, continued

- The claim has shown us that the sets $L_1 = \{a_1, a_2, \ldots, a_{\phi(n)}\}$ and $L_2 = \{aa_1, aa_2, \ldots, aa_{\phi(n)}\} \pmod{n}$ are identical sets of elements.

Proof of Euler's Theorem, continued

- The claim has shown us that the sets $L_1 = \{a_1, a_2, \dots, a_{\phi(n)}\}$ and $L_2 = \{aa_1, aa_2, \dots, aa_{\phi(n)}\} \pmod{n}$ are identical sets of elements.

- Thus $a_1 a_2 \dots a_{\phi(n)} \equiv aa_1 aa_2 \dots aa_{\phi(n)} \pmod{n}$

Proof of Euler's Theorem, continued

- The claim has shown us that the sets $L_1 = \{a_1, a_2, \ldots, a_{\phi(n)}\}$ and $L_2 = \{aa_1, aa_2, \ldots, aa_{\phi(n)}\} \pmod{n}$ are identical sets of elements.
- Thus $a_1 a_2 \ldots a_{\phi(n)} \equiv aa_1 aa_2 \ldots aa_{\phi(n)} \pmod{n}$
- $\Rightarrow a_1 a_2 \ldots a_{\phi(n)} \equiv a^{\phi(n)} a_1 a_2 \ldots a_{\phi(n)} \pmod{n}$.

Proof of Euler's Theorem, continued

- The claim has shown us that the sets $L_1 = \{a_1, a_2, \dots, a_{\phi(n)}\}$ and $L_2 = \{aa_1, aa_2, \dots, aa_{\phi(n)}\} \pmod n$ are identical sets of elements.
- Thus $a_1 a_2 \dots a_{\phi(n)} \equiv aa_1 aa_2 \dots aa_{\phi(n)} \pmod n$
- $\Rightarrow a_1 a_2 \dots a_{\phi(n)} \equiv a^{\phi(n)} a_1 a_2 \dots a_{\phi(n)} \pmod n$.
- Since each of the a_i's is relatively prime to n they can each be inverted $\pmod n$ in which case

Proof of Euler's Theorem, continued

- The claim has shown us that the sets $L_1 = \{a_1, a_2, \dots, a_{\phi(n)}\}$ and $L_2 = \{aa_1, aa_2, \dots, aa_{\phi(n)}\} \pmod{n}$ are identical sets of elements.
- Thus $a_1 a_2 \dots a_{\phi(n)} \equiv aa_1 aa_2 \dots aa_{\phi(n)} \pmod{n}$
- $\Rightarrow a_1 a_2 \dots a_{\phi(n)} \equiv a^{\phi(n)} a_1 a_2 \dots a_{\phi(n)} \pmod{n}$.
- Since each of the a_i's is relatively prime to n they can each be inverted $\pmod{n}$ in which case
- $a^{\phi(n)} \equiv 1 \pmod{n}$.

Proof of Euler's Theorem, continued

- The claim has shown us that the sets $L_1 = \{a_1, a_2, \ldots, a_{\phi(n)}\}$ and $L_2 = \{aa_1, aa_2, \ldots, aa_{\phi(n)}\} \pmod{n}$ are identical sets of elements.
- Thus $a_1 a_2 \ldots a_{\phi(n)} \equiv aa_1 aa_2 \ldots aa_{\phi(n)} \pmod{n}$
- $\Rightarrow a_1 a_2 \ldots a_{\phi(n)} \equiv a^{\phi(n)} a_1 a_2 \ldots a_{\phi(n)} \pmod{n}$.
- Since each of the a_i's is relatively prime to n they can each be inverted $\pmod{n}$ in which case
- $a^{\phi(n)} \equiv 1 \pmod{n}$.
- **QED**

Corollary to Euler's Theorem

- Suppose in the hypotheses of Euler's Theorem we let n be a prime number p.

Corollary to Euler's Theorem

- Suppose in the hypotheses of Euler's Theorem we let n be a prime number p.
- Then $\phi(p) = p - 1$. Why?

Corollary to Euler's Theorem

- Suppose in the hypotheses of Euler's Theorem we let n be a prime number p.
- Then $\phi(p) = p - 1$. Why?
- It follows from Euler's Theorem that for any prime p and any $a \in \mathbb{Z}$ such that $gcd(a, p) = 1$ we have $a^{p-1} \equiv 1 \pmod{p}$.

Corollary to Euler's Theorem

- Suppose in the hypotheses of Euler's Theorem we let n be a prime number p.
- Then $\phi(p) = p - 1$. Why?
- It follows from Euler's Theorem that for any prime p and any $a \in \mathbb{Z}$ such that $gcd(a, p) = 1$ we have $a^{p-1} \equiv 1 \pmod{p}$.
- This special case of Euler's Theorem is due to Fermat and is called Fermat's Little Theorem, not to be confused with Fermat's *Last* Theorem.

Back to primality testing algorithms...

A Better Method: Randomization

- **Algorithm Lehann's Primality Test (1982)**
- **Input.** $n \geq 3$; n odd and $\ell \geq 2$ with $b[1..\ell]$ a $1 \times \ell$ array.
- **Output.** *Is n prime? Can we tell?*
- **for** $i = 1$ **to** ℓ **do**
- a is a randomly chosen element from $\{1, 2, \ldots, n\}$
- $c = a^{\frac{n-1}{2}} \pmod{n}$
- **if** $c \notin \{1, n-1\}$
- **then return** 1
- **else** $b[i] = c$
- **if** $b[1] = b[2] = \cdots = b[\ell] = 1$ (*)
- **then return** 1
- **else return** 0

Analysis of Lehmann's Primality Test

- What exactly is the output??

Analysis of Lehmann's Primality Test

- What exactly is the output??
- If n is actually prime then study $(a^{\frac{n-1}{2}})^2 \equiv c^2 \pmod{n} \Rightarrow a^{n-1} \equiv c^2 \pmod{n}$.

Analysis of Lehmann's Primality Test

- What exactly is the output??
- If n is actually prime then study $(a^{\frac{n-1}{2}})^2 \equiv c^2 \pmod{n} \Rightarrow a^{n-1} \equiv c^2 \pmod{n}$.
- By Fermat's Little Theorem, we have that $c \in \{1, -1\} \equiv \{1, n-1\} \pmod{n}$.

Analysis of Lehmann's Primality Test

- What exactly is the output??
- If n is actually prime then study $(a^{\frac{n-1}{2}})^2 \equiv c^2 \pmod{n} \Rightarrow a^{n-1} \equiv c^2 \pmod{n}$.
- By Fermat's Little Theorem, we have that $c \in \{1, -1\} \equiv \{1, n-1\} \pmod{n}$.
- Thus if c turns out to be anything other than 1 or -1 $\pmod{n}$, then n cannot be prime.

Analysis of Lehmann's Primality Test

- What exactly is the output??
- If n is actually prime then study $(a^{\frac{n-1}{2}})^2 \equiv c^2 \pmod{n} \Rightarrow a^{n-1} \equiv c^2 \pmod{n}$.
- By Fermat's Little Theorem, we have that $c \in \{1, -1\} \equiv \{1, n-1\} \pmod{n}$.
- Thus if c turns out to be anything other than 1 or -1 $\pmod{n}$, then n cannot be prime.
- In line (*) we will have some entry $\notin \{1, n-1\}$

Analysis of Lehmann's Primality Test

- What exactly is the output??
- If n is actually prime then study $(a^{\frac{n-1}{2}})^2 \equiv c^2 \pmod{n} \Rightarrow a^{n-1} \equiv c^2 \pmod{n}$.
- By Fermat's Little Theorem, we have that $c \in \{1, -1\} \equiv \{1, n-1\} \pmod{n}$.
- Thus if c turns out to be anything other than 1 or -1 $\pmod{n}$, then n cannot be prime.
- In line (*) we will have some entry $\notin \{1, n-1\}$
- and can conclude that n is not prime since Fermat's Little Theorem is not satisfied.

Lehmann's Primality Test, continued

- We can run this random experiment ℓ times for ℓ as large as we want.

Lehmann's Primality Test, continued

- We can run this random experiment ℓ times for ℓ as large as we want.
- The output array **b** is called a **random variable**.

Lehmann's Primality Test, continued

- We can run this random experiment ℓ times for ℓ as large as we want.
- The output array **b** is called a **random variable**.
- **Question.** What is the probability that we will get an incorrect answer as output?

Lehmann's Primality Test, continued

- We can run this random experiment ℓ times for ℓ as large as we want.
- The output array **b** is called a **random variable**.
- **Question.** What is the probability that we will get an incorrect answer as output?
- **Analysis.**

Lehmann's Primality Test, continued

- We can run this random experiment ℓ times for ℓ as large as we want.
- The output array **b** is called a **random variable**.
- **Question.** What is the probability that we will get an incorrect answer as output?
- **Analysis.**
- **Case 1.** The integer n is actually a prime number and thus we hope that the output is "0".

Lehmann's Primality Test, continued

- We can run this random experiment ℓ times for ℓ as large as we want.
- The output array **b** is called a **random variable**.
- **Question.** What is the probability that we will get an incorrect answer as output?
- **Analysis.**
- **Case 1.** The integer n is actually a prime number and thus we hope that the output is "0".
- **Fact.** When n is an odd prime, exactly half of $\{1, 2, \ldots, n\}$ satisfy $a^{\frac{n-1}{2}} \equiv 1 \pmod{n}$ and the other half satisfy $a^{\frac{n-1}{2}} \equiv n-1 \pmod{n}$ (*take number theory in the spring to see why...*)

- The loop will run through the full ℓ rounds;

- The loop will run through the full ℓ rounds;
- the probability that $c_1 = c_2 = \cdots = c_\ell = 1$ and that the wrong output is produced is $2^{-\ell}$ (probability $= \frac{1}{2}$ for each c_i).

- The loop will run through the full ℓ rounds;
- the probability that $c_1 = c_2 = \cdots = c_\ell = 1$ and that the wrong output is produced is $2^{-\ell}$ (probability $= \frac{1}{2}$ for each c_i).
- **Case 2.** The integer n is composite. We hope the output is "1".

- The loop will run through the full ℓ rounds;
- the probability that $c_1 = c_2 = \cdots = c_\ell = 1$ and that the wrong output is produced is $2^{-\ell}$ (probability $= \frac{1}{2}$ for each c_i).
- **Case 2.** The integer n is composite. We hope the output is "1".
- **Subcase 2.1.** If there is no $a \in \{1, 2, \ldots, n\}$ such that $a^{\frac{n-1}{2}} \equiv n - 1 \pmod{n}$ then

- The loop will run through the full ℓ rounds;
- the probability that $c_1 = c_2 = \cdots = c_\ell = 1$ and that the wrong output is produced is $2^{-\ell}$ (probability $= \frac{1}{2}$ for each c_i).
- **Case 2.** The integer n is composite. We hope the output is "1".
- **Subcase 2.1.** If there is no $a \in \{1, 2, \ldots, n\}$ such that $a^{\frac{n-1}{2}} \equiv n - 1 \pmod{n}$ then
- the output of the algorithm will be "1", which is correct.

- The loop will run through the full ℓ rounds;
- the probability that $c_1 = c_2 = \cdots = c_\ell = 1$ and that the wrong output is produced is $2^{-\ell}$ (probability $= \frac{1}{2}$ for each c_i).
- **Case 2.** The integer n is composite. We hope the output is "1".
- **Subcase 2.1.** If there is no $a \in \{1, 2, \ldots, n\}$ such that $a^{\frac{n-1}{2}} \equiv n - 1 \pmod{n}$ then
- the output of the algorithm will be "1", which is correct.
- **Subcase 2.2.** If there is some $a \in \{1, 2, \ldots, n\}$ such that $a^{\frac{n-1}{2}} \equiv n - 1 \pmod{n}$ then

- The loop will run through the full ℓ rounds;
- the probability that $c_1 = c_2 = \cdots = c_\ell = 1$ and that the wrong output is produced is $2^{-\ell}$ (probability $= \frac{1}{2}$ for each c_i).
- **Case 2.** The integer n is composite. We hope the output is "1".
- **Subcase 2.1.** If there is no $a \in \{1, 2, \ldots, n\}$ such that $a^{\frac{n-1}{2}} \equiv n-1 \pmod{n}$ then
- the output of the algorithm will be "1", which is correct.
- **Subcase 2.2.** If there is some $a \in \{1, 2, \ldots, n\}$ such that $a^{\frac{n-1}{2}} \equiv n-1 \pmod{n}$ then
- more than half of the elements in $\{1, 2, \ldots, , n\}$ satisfy $a^{\frac{n-1}{2}} \not\equiv n-1 \pmod{n}$

- The loop will run through the full ℓ rounds;
- the probability that $c_1 = c_2 = \cdots = c_\ell = 1$ and that the wrong output is produced is $2^{-\ell}$ (probability $= \frac{1}{2}$ for each c_i).
- **Case 2.** The integer n is composite. We hope the output is "1".
- **Subcase 2.1.** If there is no $a \in \{1, 2, \ldots, n\}$ such that $a^{\frac{n-1}{2}} \equiv n - 1 \pmod{n}$ then
- the output of the algorithm will be "1", which is correct.
- **Subcase 2.2.** If there is some $a \in \{1, 2, \ldots, n\}$ such that $a^{\frac{n-1}{2}} \equiv n - 1 \pmod{n}$ then
- more than half of the elements in $\{1, 2, \ldots, , n\}$ satisfy $a^{\frac{n-1}{2}} \not\equiv n - 1 \pmod{n}$
- then more than half of the elements in $\{1, 2, \ldots, n\}$ satisfy $a^{\frac{n-1}{2}} \not\equiv n - 1$ or $1 \pmod{n}$.

Lehmann's Primality Test, continued

- Thus the probability that the loop runs for ℓ rounds is no more than $2^{-\ell}$.

Lehmann's Primality Test, continued

- Thus the probability that the loop runs for ℓ rounds is no more than $2^{-\ell}$.
- All in all, that an output of "0" is given is no more than $2^{-\ell}$.

Lehmann's Primality Test, continued

- Thus the probability that the loop runs for ℓ rounds is no more than $2^{-\ell}$.
- All in all, that an output of "0" is given is no more than $2^{-\ell}$.
- The same probability of an incorrect output was achieved in both cases, and hence applies to the algorithm as a whole.

Lehmann's Primality Test, continued

- Thus the probability that the loop runs for ℓ rounds is no more than $2^{-\ell}$.
- All in all, that an output of "0" is given is no more than $2^{-\ell}$.
- The same probability of an incorrect output was achieved in both cases, and hence applies to the algorithm as a whole.
- Note that increasing the size of ℓ decreases the probability that the incorrect output is given.

Lehmann's Primality Test, continued

- Thus the probability that the loop runs for ℓ rounds is no more than $2^{-\ell}$.
- All in all, that an output of "0" is given is no more than $2^{-\ell}$.
- The same probability of an incorrect output was achieved in both cases, and hence applies to the algorithm as a whole.
- Note that increasing the size of ℓ decreases the probability that the incorrect output is given.
- **Computational Complexity.** The heaviest cost is in computing $a^{\frac{n-1}{2}}$.

Lehmann's Primality Test, continued

- Thus the probability that the loop runs for ℓ rounds is no more than $2^{-\ell}$.
- All in all, that an output of "0" is given is no more than $2^{-\ell}$.
- The same probability of an incorrect output was achieved in both cases, and hence applies to the algorithm as a whole.
- Note that increasing the size of ℓ decreases the probability that the incorrect output is given.
- **Computational Complexity.** The heaviest cost is in computing $a^{\frac{n-1}{2}}$.
- Using fast exponentiation (ie, repeated squaring) leads to at most $2 log n$ multiplications and divisions of integers that are $\leq n^2$.

Next week...

We'll study the RSA public encryption algorithm.