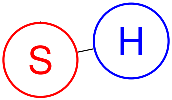
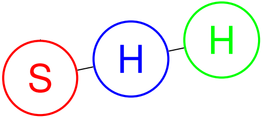
# CS 170: Algorithms

# CS 170: Algorithms

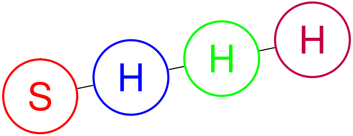
# CS 170: Algorithms



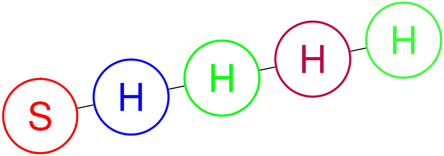
# CS 170: Algorithms



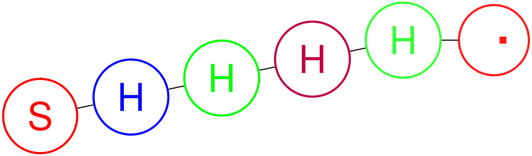
# CS 170: Algorithms



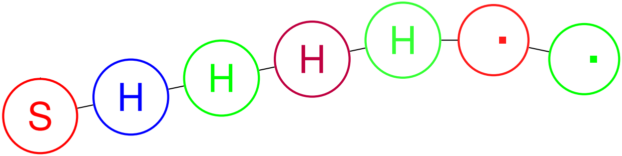
# CS 170: Algorithms



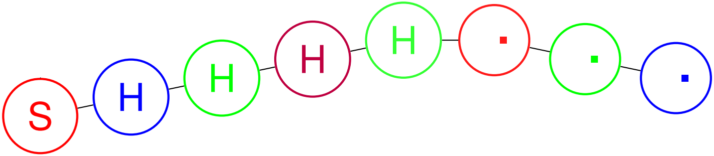
# CS 170: Algorithms



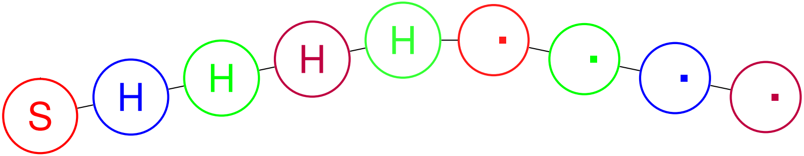
CS 170: Algorithms



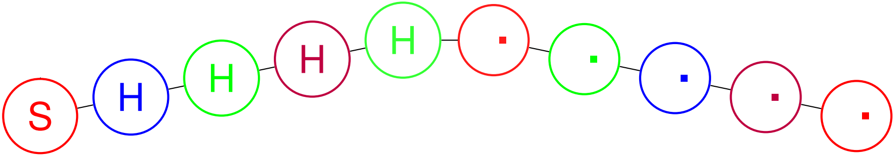
CS 170: Algorithms



# CS 170: Algorithms



# CS 170: Algorithms



# Lecture in a minute.

FFT Wrapup:

# Lecture in a minute.

FFT Wrapup:

Evaluate degree $n$ polynomial on $n$ points.

Recursion:

Evaluate Odd/Even polynomials on squares of points.

Which $n = 2^k$ points?

# Lecture in a minute.

FFT Wrapup:

Evaluate degree $n$ polynomial on $n$ points.

Recursion:

Evaluate Odd/Even polynomials on squares of points.

Which $n = 2^k$ points?

$n$th root of unity: complex numbers.

"squares of $n$th root of unity are $n/2$th roots of unity."

Evaluate Odd/Even polynomials on $n/2$ points. $\implies$

$T(n) = 2T(n/2) + O(n) = O(n \log n)$.

# Lecture in a minute.

FFT Wrapup:

Evaluate degree $n$ polynomial on $n$ points.

Recursion:

Evaluate Odd/Even polynomials on squares of points.

Which $n = 2^k$ points?

$n$th root of unity: complex numbers.

"squares of $n$th root of unity are $n/2$th roots of unity."

Evaluate Odd/Even polynomials on $n/2$ points. $\implies$

$T(n) = 2T(n/2) + O(n) = O(n \log n)$.

Graphs:

# Lecture in a minute.

FFT Wrapup:

Evaluate degree $n$ polynomial on $n$ points.

Recursion:

Evaluate Odd/Even polynomials on squares of points.

Which $n = 2^k$ points?

$n$th root of unity: complex numbers.

"squares of $n$th root of unity are $n/2$th roots of unity."

Evaluate Odd/Even polynomials on $n/2$ points. $\implies$

$T(n) = 2T(n/2) + O(n) = O(n \log n)$.

Graphs:

$G = (V, E)$, $V$ - vertices, $E$ edges.

Representations:

Matrix: $|V|^2$ space, fast check for edges.

Adjacency List: $O(|V| + |E|)$ space. More complicated.

# Lecture in a minute.

FFT Wrapup:

Evaluate degree $n$ polynomial on $n$ points.

Recursion:

Evaluate Odd/Even polynomials on squares of points.

Which $n = 2^k$ points?

$n$th root of unity: complex numbers.

"squares of $n$th root of unity are $n/2$th roots of unity."

Evaluate Odd/Even polynomials on $n/2$ points. $\implies$

$T(n) = 2T(n/2) + O(n) = O(n \log n)$.

Graphs:

$G = (V, E)$, $V$ - vertices, $E$ edges.

Representations:

Matrix: $|V|^2$ space, fast check for edges.

Adjacency List: $O(|V| + |E|)$ space. More complicated.

Procedure: explore($v$).

Explores the graph.

Uses a stack.

Nonrecursive non-loop counting Runtime analysis.

Every little move she makes...

# Fast Fourier Transform.

Evaluate degree $n$ polynomial $A(x)$ on $n$ points.

# Fast Fourier Transform.

Evaluate degree $n$ polynomial $A(x)$ on $n$ points.

Ideas:

# Fast Fourier Transform.

Evaluate degree $n$ polynomial $A(x)$ on $n$ points.

Ideas:

# Fast Fourier Transform.

Evaluate degree $n$ polynomial $A(x)$ on $n$ points.

Ideas:

1) $A(x) = A_e(x^2) + x(A_o(x^2))$.

# Fast Fourier Transform.

Evaluate degree $n$ polynomial $A(x)$ on $n$ points.

Ideas:

1) $A(x) = A_e(x^2) + x(A_o(x^2))$.

$A_e(x) = a_0 + a_2 x + a_4 x^2 \dots$.

# Fast Fourier Transform.

Evaluate degree $n$ polynomial $A(x)$ on $n$ points.

Ideas:

1) $A(x) = A_e(x^2) + x(A_o(x^2))$.

$A_e(x) = a_0 + a_2 x + a_4 x^2 \ldots.$

$A_o(x) = a_1 + a_3 x + a_5 x^2 \ldots.$

# Fast Fourier Transform.

Evaluate degree $n$ polynomial $A(x)$ on $n$ points.

Ideas:

1) $A(x) = A_e(x^2) + x(A_o(x^2))$.

$A_e(x) = a_0 + a_2 x + a_4 x^2 ....$

$A_o(x) = a_1 + a_3 x + a_5 x^2 ....$

2) Roots of Unity.

# Fast Fourier Transform.

Evaluate degree $n$ polynomial $A(x)$ on $n$ points.

Ideas:

1) $A(x) = A_e(x^2) + x(A_o(x^2))$.

$A_e(x) = a_0 + a_2 x + a_4 x^2 ....$

$A_o(x) = a_1 + a_3 x + a_5 x^2 ....$

2) Roots of Unity.

(a) $n = 2^k$

# Fast Fourier Transform.

Evaluate degree $n$ polynomial $A(x)$ on $n$ points.

Ideas:

1) $A(x) = A_e(x^2) + x(A_o(x^2))$.

$A_e(x) = a_0 + a_2 x + a_4 x^2 ....$

$A_o(x) = a_1 + a_3 x + a_5 x^2 ....$

2) Roots of Unity.

(a) $n = 2^k$

(b) points are $1, \omega, \omega^2, \ldots, \omega^{n-1}$

# Fast Fourier Transform.

Evaluate degree $n$ polynomial $A(x)$ on $n$ points.

Ideas:

1) $A(x) = A_e(x^2) + x(A_o(x^2))$.

$A_e(x) = a_0 + a_2x + a_4x^2 ....$

$A_o(x) = a_1 + a_3x + a_5x^2 ....$

2) Roots of Unity.

(a) $n = 2^k$

(b) points are $1, \omega, \omega^2, \ldots, \omega^{n-1}$

where $\omega$ is primitive $n$th root of unity: $\omega^n = 1$.

# Evaluation of polynomial: Proof of recursive formula.

$$A(x) = A_e(x^2) + x(A_o(x^2))$$

# Evaluation of polynomial: Proof of recursive formula.

$$A(x) = A_e(x^2) + x(A_o(x^2))$$

where

$$A(x) = a_0 x^0 + \ldots a_d x^d \ldots$$

# Evaluation of polynomial: Proof of recursive formula.

$$A(x) = A_e(x^2) + x(A_o(x^2))$$

where

$$A(x) = a_0 x^0 + \ldots a_d x^d \ldots$$

Even coefficient polynomial.

# Evaluation of polynomial: Proof of recursive formula.

$$A(x) = A_e(x^2) + x(A_o(x^2))$$

where

$$A(x) = a_0 x^0 + \dots a_d x^d \dots$$

Even coefficient polynomial.

$$A_e(y) = a_0 + a_2 x + a_4 y^2 ... a_{2d} y^d ....$$

# Evaluation of polynomial: Proof of recursive formula.

$$A(x) = A_e(x^2) + x(A_o(x^2))$$

where

$$A(x) = a_0 x^0 + \ldots a_d x^d \ldots$$

Even coefficient polynomial.

$$A_e(y) = a_0 + a_2 x + a_4 y^2 ... a_{2d} y^d ....$$

Odd coefficient polynomial.

# Evaluation of polynomial: Proof of recursive formula.

$$A(x) = A_e(x^2) + x(A_o(x^2))$$

where

$$A(x) = a_0 x^0 + \ldots a_d x^d \ldots$$

Even coefficient polynomial.

$$A_e(y) = a_0 + a_2 x + a_4 y^2 ... a_{2d} y^d ....$$

Odd coefficient polynomial.

$$A_o(y) = a_1 + a_3 y + a_5 y^2 ....$$

# Evaluation of polynomial: Proof of recursive formula.

$$A(x) = A_e(x^2) + x(A_o(x^2))$$

where

$$A(x) = a_0 x^0 + \ldots a_d x^d \ldots$$

Even coefficient polynomial.

$$A_e(y) = a_0 + a_2 x + a_4 y^2 ... a_{2d} y^d ....$$

Odd coefficient polynomial.

$$A_o(y) = a_1 + a_3 y + a_5 y^2 ....$$

Proof:

Should get $a_d x^d$ in $A_e(x^2) + x(A_o(x^2))$.

# Evaluation of polynomial: Proof of recursive formula.

$$A(x) = A_e(x^2) + x(A_o(x^2))$$

where

$$A(x) = a_0 x^0 + \ldots a_d x^d \ldots$$

Even coefficient polynomial.

$$A_e(y) = a_0 + a_2 x + a_4 y^2 ... a_{2d} y^d ....$$

Odd coefficient polynomial.

$$A_o(y) = a_1 + a_3 y + a_5 y^2 ....$$

Proof:

Should get $a_d x^d$ in $A_e(x^2) + x(A_o(x^2))$.

$d$ is even:

# Evaluation of polynomial: Proof of recursive formula.

$$A(x) = A_e(x^2) + x(A_o(x^2))$$

where

$$A(x) = a_0x^0 + \ldots a_dx^d \ldots$$

Even coefficient polynomial.

$$A_e(y) = a_0 + a_2x + a_4y^2 ... a_{2d}y^d ....$$

Odd coefficient polynomial.

$$A_o(y) = a_1 + a_3y + a_5y^2 ....$$

Proof:

Should get $a_dx^d$ in $A_e(x^2) + x(A_o(x^2))$.

$d$ is even:

$d/2$th coefficient of $A_e(y)$ is $a_d$.

# Evaluation of polynomial: Proof of recursive formula.

$$A(x) = A_e(x^2) + x(A_o(x^2))$$

where

$$A(x) = a_0 x^0 + \ldots a_d x^d \ldots$$

Even coefficient polynomial.

$$A_e(y) = a_0 + a_2 x + a_4 y^2 ... a_{2d} y^d ....$$

Odd coefficient polynomial.

$$A_o(y) = a_1 + a_3 y + a_5 y^2 ....$$

Proof:

Should get $a_d x^d$ in $A_e(x^2) + x(A_o(x^2))$.

$d$ is even:

$d/2$th coefficient of $A_e(y)$ is $a_d$.

when $y = x^2$, $a_d y^{d/2} = a_d (x^2)^{d/2} = a_d x^d$

# Evaluation of polynomial: Proof of recursive formula.

$$A(x) = A_e(x^2) + x(A_o(x^2))$$

where

$$A(x) = a_0x^0 + \ldots a_dx^d \ldots$$

Even coefficient polynomial.

$$A_e(y) = a_0 + a_2x + a_4y^2 ... a_{2d}y^d ....$$

Odd coefficient polynomial.

$$A_o(y) = a_1 + a_3y + a_5y^2 ....$$

Proof:
Should get $a_dx^d$ in $A_e(x^2) + x(A_o(x^2))$.

$d$ is even:
$d/2$th coefficient of $A_e(y)$ is $a_d$.
when $y = x^2$, $a_dy^{d/2} = a_d(x^2)^{d/2} = a_dx^d$
$d$ is odd:

# Evaluation of polynomial: Proof of recursive formula.

$$A(x) = A_e(x^2) + x(A_o(x^2))$$

where

$$A(x) = a_0 x^0 + \ldots a_d x^d \ldots$$

Even coefficient polynomial.

$$A_e(y) = a_0 + a_2 x + a_4 y^2 ... a_{2d} y^d ....$$

Odd coefficient polynomial.

$$A_o(y) = a_1 + a_3 y + a_5 y^2 ....$$

Proof:
Should get $a_d x^d$ in $A_e(x^2) + x(A_o(x^2))$.

$d$ is even:
$d/2$th coefficient of $A_e(y)$ is $a_d$.
when $y = x^2$, $a_d y^{d/2} = a_d (x^2)^{d/2} = a_d x^d$
$d$ is odd:
$i = \frac{(d-1)}{2}$th coefficient of $A_o(y)$ is $a_d$.

# Evaluation of polynomial: Proof of recursive formula.

$$A(x) = A_e(x^2) + x(A_o(x^2))$$

where

$$A(x) = a_0 x^0 + \ldots a_d x^d \ldots$$

Even coefficient polynomial.

$$A_e(y) = a_0 + a_2 x + a_4 y^2 ... a_{2d} y^d ....$$

Odd coefficient polynomial.

$$A_o(y) = a_1 + a_3 y + a_5 y^2 ....$$

Proof:

Should get $a_d x^d$ in $A_e(x^2) + x(A_o(x^2))$.

$d$ is even:

$d/2$th coefficient of $A_e(y)$ is $a_d$.

when $y = x^2$, $a_d y^{d/2} = a_d (x^2)^{d/2} = a_d x^d$

$d$ is odd:

$i = \frac{(d-1)}{2}$th coefficient of $A_o(y)$ is $a_d$.

When $y = x^2$, $A_o(y)$ contains $a_d y^{(d-1)/2}$

# Evaluation of polynomial: Proof of recursive formula.

$$A(x) = A_e(x^2) + x(A_o(x^2))$$

where

$$A(x) = a_0 x^0 + \ldots a_d x^d \ldots$$

Even coefficient polynomial.

$$A_e(y) = a_0 + a_2 x + a_4 y^2 \ldots a_{2d} y^d \ldots.$$

Odd coefficient polynomial.

$$A_o(y) = a_1 + a_3 y + a_5 y^2 \ldots.$$

Proof:
Should get $a_d x^d$ in $A_e(x^2) + x(A_o(x^2))$.

$d$ is even:
$d/2$th coefficient of $A_e(y)$ is $a_d$.
when $y = x^2$, $a_d y^{d/2} = a_d (x^2)^{d/2} = a_d x^d$
$d$ is odd:
$i = \frac{(d-1)}{2}$th coefficient of $A_o(y)$ is $a_d$.
When $y = x^2$, $A_o(y)$ contains $a_d y^{(d-1)/2} = a_d (x^{d-1})$

# Evaluation of polynomial: Proof of recursive formula.

$$A(x) = A_e(x^2) + x(A_o(x^2))$$

where

$A(x) = a_0x^0 + \ldots a_dx^d \ldots$

Even coefficient polynomial.

$A_e(y) = a_0 + a_2x + a_4y^2 \ldots a_{2d}y^d \ldots.$

Odd coefficient polynomial.

$A_o(y) = a_1 + a_3y + a_5y^2 \ldots.$

Proof:

Should get $a_dx^d$ in $A_e(x^2) + x(A_o(x^2))$.

$d$ is even:

$d/2$th coefficient of $A_e(y)$ is $a_d$.

when $y = x^2$, $a_dy^{d/2} = a_d(x^2)^{d/2} = a_dx^d$

$d$ is odd:

$i = \frac{(d-1)}{2}$th coefficient of $A_o(y)$ is $a_d$.

When $y = x^2$, $A_o(y)$ contains $a_dy^{(d-1)/2} = a_d(x^{d-1})$

$\implies xA_o(x^2)$ contains $a_dx^d$.

# Evaluation of polynomial: Proof of recursive formula.

$A(x) = A_e(x^2) + x(A_o(x^2))$

where

$A(x) = a_0 x^0 + \ldots a_d x^d \ldots$

Even coefficient polynomial.

$A_e(y) = a_0 + a_2 x + a_4 y^2 ... a_{2d} y^d ....$

Odd coefficient polynomial.

$A_o(y) = a_1 + a_3 y + a_5 y^2 ....$

Proof:

Should get $a_d x^d$ in $A_e(x^2) + x(A_o(x^2))$.

$d$ is even:

$d/2$th coefficient of $A_e(y)$ is $a_d$.

when $y = x^2$, $a_d y^{d/2} = a_d (x^2)^{d/2} = a_d x^d$

$d$ is odd:

$i = \frac{(d-1)}{2}$th coefficient of $A_o(y)$ is $a_d$.

When $y = x^2$, $A_o(y)$ contains $a_d y^{(d-1)/2} = a_d (x^{d-1})$

$\implies xA_o(x^2)$ contains $a_d x^d$.

# Multiplying Complex Numbers

Complex numbers:

# Multiplying Complex Numbers

Complex numbers:

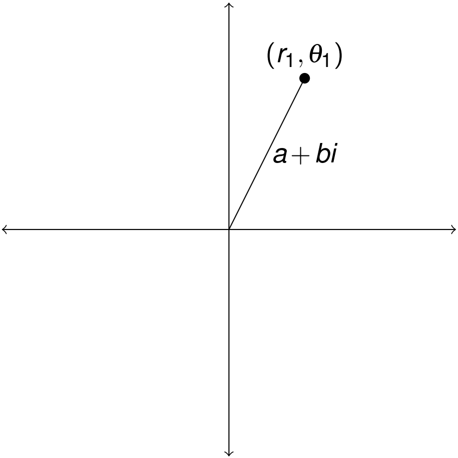
Coordinate representation: $a + bi$

# Multiplying Complex Numbers

Complex numbers:

Coordinate representation: $a+bi$

Polar representation: $(r,\theta)$ where $r=\sqrt{a^2+b^2}$ and $\theta=\tan^{-1}\frac{b}{a}$.



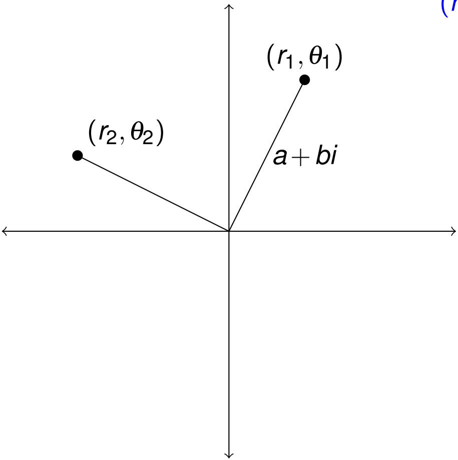
# Multiplying Complex Numbers

Complex numbers:

Coordinate representation: $a+bi$

Polar representation: $(r,\theta)$ where $r=\sqrt{a^2+b^2}$ and $\theta=\tan^{-1}\frac{b}{a}$.

$(r_1,\theta_1)\times(r_2,\theta_2)$

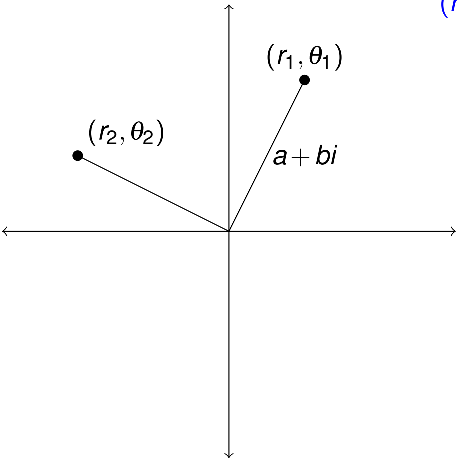


# Multiplying Complex Numbers

Complex numbers:

Coordinate representation: $a + bi$

Polar representation: $(r, \theta)$ where $r = \sqrt{a^2 + b^2}$ and $\theta = \tan^{-1} \frac{b}{a}$.




$(r_1, \theta_1) \times (r_2, \theta_2)$

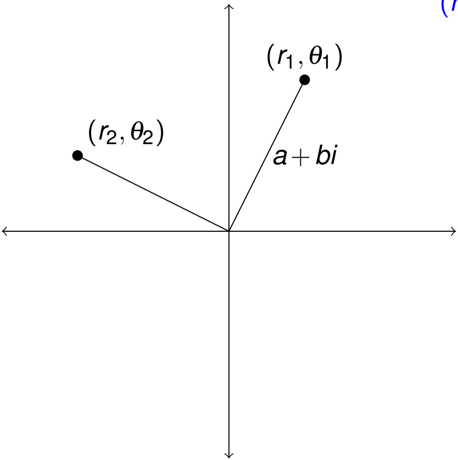
$= r_1 e^{i\theta_1} \times r_2 e^{i\theta_2}$

# Multiplying Complex Numbers

Complex numbers:

Coordinate representation: $a + bi$

Polar representation: $(r, \theta)$ where $r = \sqrt{a^2 + b^2}$ and $\theta = \tan^{-1} \frac{b}{a}$.




$(r_1, \theta_1) \times (r_2, \theta_2)$

$$= r_1 e^{i\theta_1} \times r_2 e^{i\theta_2}$$

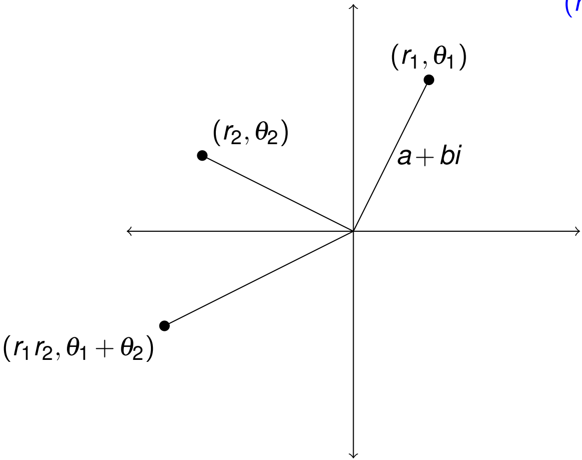
$$= (r_1 r_2) e^{i(\theta_1 + \theta_2)}$$

# Multiplying Complex Numbers

Complex numbers:

Coordinate representation: $a + bi$

Polar representation: $(r, \theta)$ where $r = \sqrt{a^2 + b^2}$ and $\theta = \tan^{-1} \frac{b}{a}$.



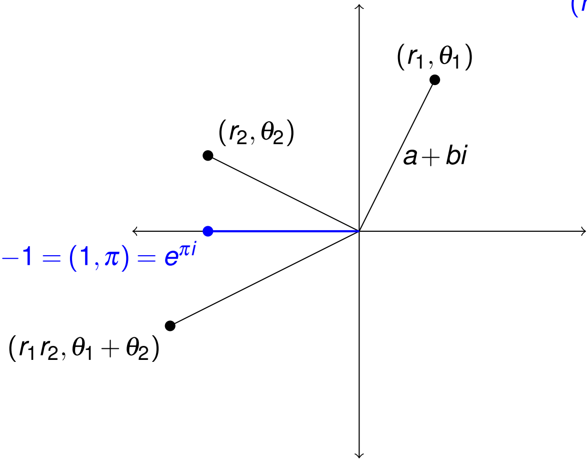

$$
\begin{aligned}
&(r_1, \theta_1) \times (r_2, \theta_2) \\
&\quad = r_1 e^{i\theta_1} \times r_2 e^{i\theta_2} \\
&\quad = (r_1 r_2) e^{i(\theta_1 + \theta_2)} \\
&\quad = (r_1 r_2, \theta_1 + \theta_2)
\end{aligned}
$$

# Multiplying Complex Numbers

Complex numbers:

Coordinate representation: $a + bi$

Polar representation: $(r, \theta)$ where $r = \sqrt{a^2 + b^2}$ and $\theta = \tan^{-1} \frac{b}{a}$.




$$
\begin{aligned}
&(r_1, \theta_1) \times (r_2, \theta_2) \\
&\quad = r_1 e^{i\theta_1} \times r_2 e^{i\theta_2} \\
&\quad = (r_1 r_2) e^{i(\theta_1 + \theta_2)} \\
&\quad = (r_1 r_2, \theta_1 + \theta_2)
\end{aligned}
$$

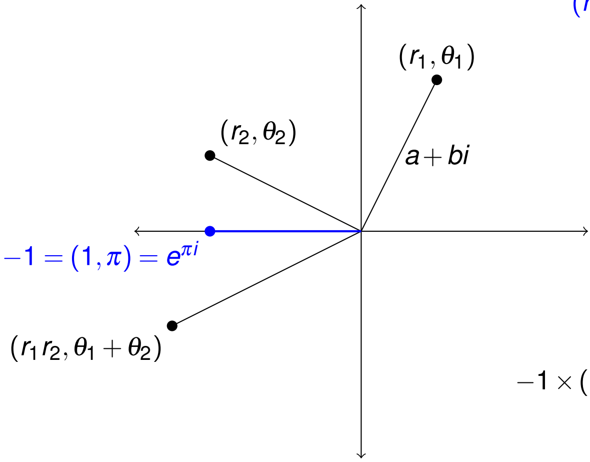
# Multiplying Complex Numbers

Complex numbers:

Coordinate representation: $a + bi$

Polar representation: $(r, \theta)$ where $r = \sqrt{a^2 + b^2}$ and $\theta = \tan^{-1} \frac{b}{a}$.

$$(r_1, \theta_1) \times (r_2, \theta_2)$$
$$= r_1 e^{i\theta_1} \times r_2 e^{i\theta_2}$$
$$= (r_1 r_2) e^{i(\theta_1 + \theta_2)}$$
$$= (r_1 r_2, \theta_1 + \theta_2)$$




$$-1 \times (r_1, \theta_1) = (r_1, \theta_1 + \pi)$$

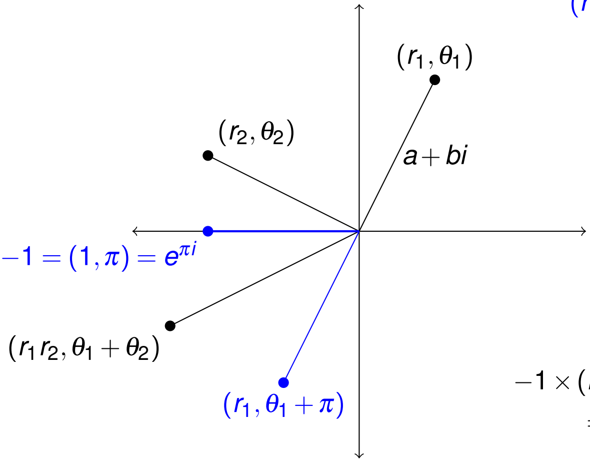
# Multiplying Complex Numbers

Complex numbers:

Coordinate representation: $a + bi$

Polar representation: $(r, \theta)$ where $r = \sqrt{a^2 + b^2}$ and $\theta = \tan^{-1} \frac{b}{a}$.

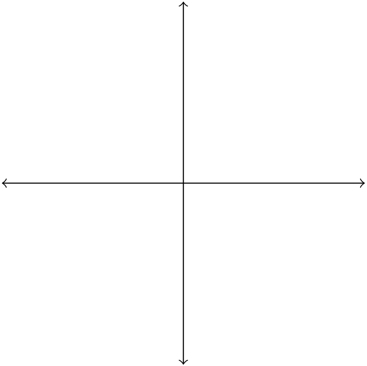
$$\begin{aligned}
&(r_1, \theta_1) \times (r_2, \theta_2) \\
&\quad = r_1 e^{i\theta_1} \times r_2 e^{i\theta_2} \\
&\quad = (r_1 r_2) e^{i(\theta_1 + \theta_2)} \\
&\quad = (r_1 r_2, \theta_1 + \theta_2)
\end{aligned}$$




$$\begin{aligned}
-1 \times (r_1, \theta_1) &= (r_1, \theta_1 + \pi) \\
&= r_1 e^{r_1(\theta_1 + \pi)i}
\end{aligned}$$

# The $n$th complex roots of unity.

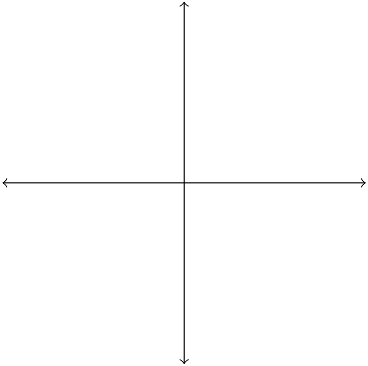
Solutions to $z^n = 1$



# The $n$th complex roots of unity.

Solutions to $z^n = 1$

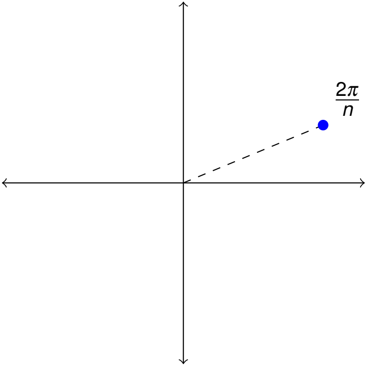
$$(1, \tfrac{2\pi}{n})^n = (1, \tfrac{2\pi}{n} \times n) = (1, 2\pi) = 1!$$



# The $n$th complex roots of unity.

Solutions to $z^n = 1$

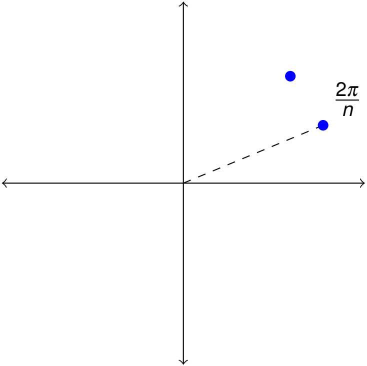
$$(1, \tfrac{2\pi}{n})^n = (1, \tfrac{2\pi}{n} \times n) = (1, 2\pi) = 1!$$



# The $n$th complex roots of unity.

Solutions to $z^n = 1$

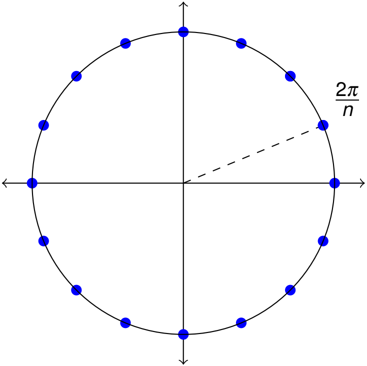
$$(1, \frac{4\pi}{n})^n = (1, \frac{4\pi}{n} \times n) = (1, 4\pi) = 1!$$



# The $n$th complex roots of unity.

Solutions to $z^n = 1$

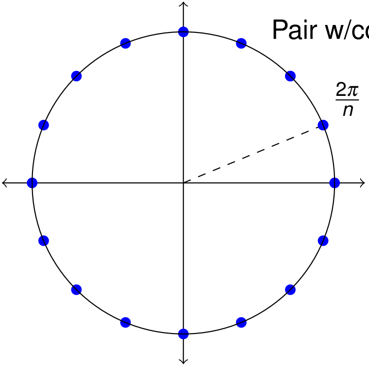
$$(1, \tfrac{2k\pi}{n})^n = (1, \tfrac{2k\pi}{n} \times n) = (1, 2k\pi) = 1!$$



# The $n$th complex roots of unity.

Solutions to $z^n = 1$

$$(1, \theta + \pi)^2 = (1, 2\theta + 2\pi) = \boxed{(1, 2\theta)} = (1, \theta)^2.$$

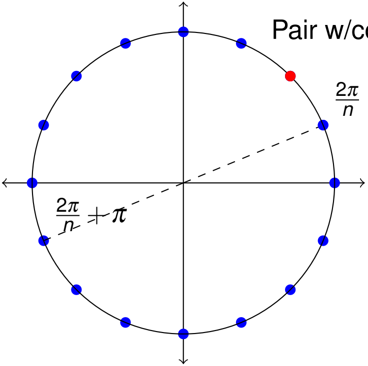



Pair w/common square!

# The $n$th complex roots of unity.

Solutions to $z^n = 1$

$$(1, \theta + \pi)^2 = (1, 2\theta + 2\pi) = \boxed{(1, 2\theta)} = (1, \theta)^2.$$

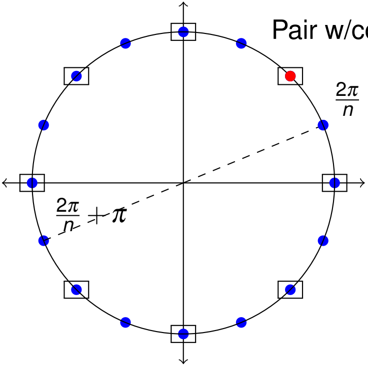



Pair w/common square!

# The $n$th complex roots of unity.

Solutions to $z^n = 1$

$$(1, \theta + \pi)^2 = (1, 2\theta + 2\pi) = \boxed{(1, 2\theta)} = (1, \theta)^2.$$

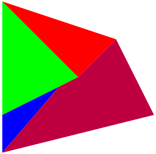


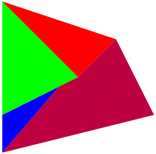

Pair w/common square!

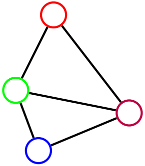
Squares: $\frac{n}{2}$th roots.

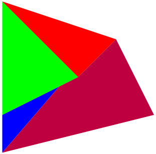
# Next

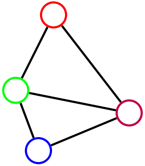
1. Graphs
2. Reachability.



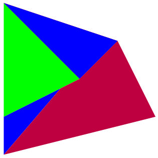
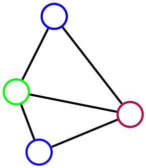




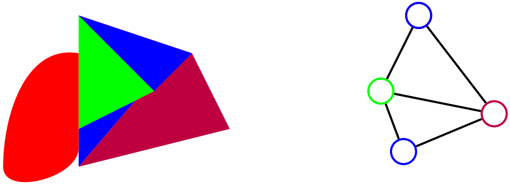


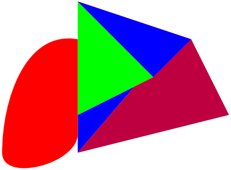


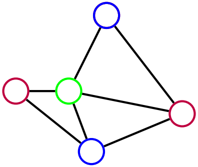
Fewer Colors?

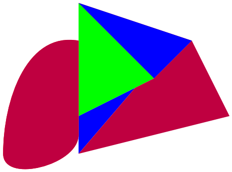
 

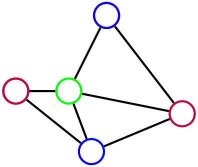
Yes! Three colors.

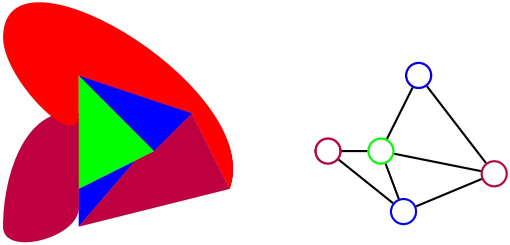


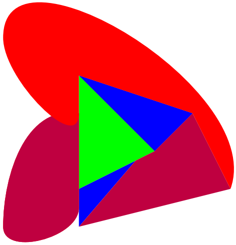


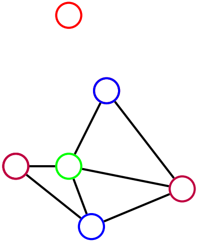


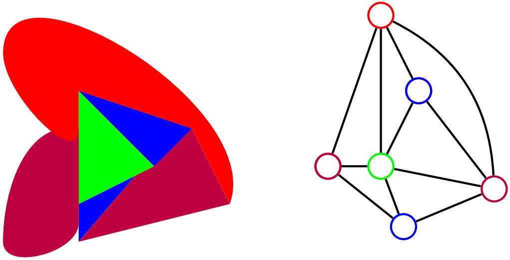


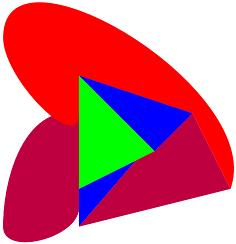


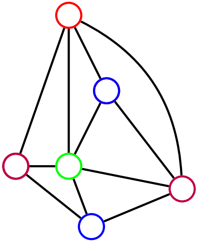




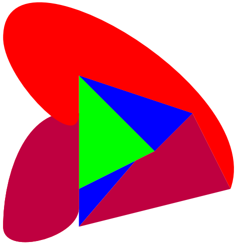


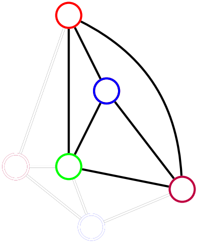


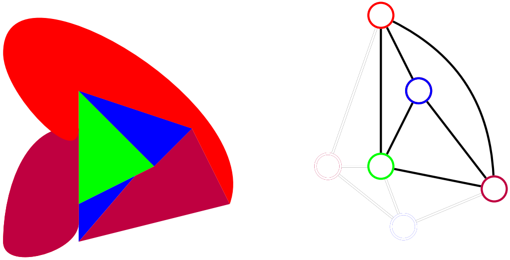




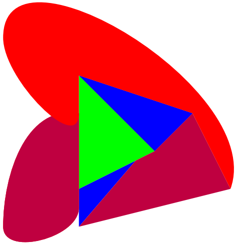
Fewer Colors?

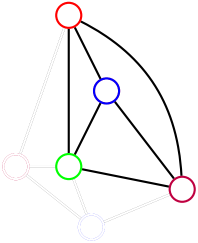






Four colors required!

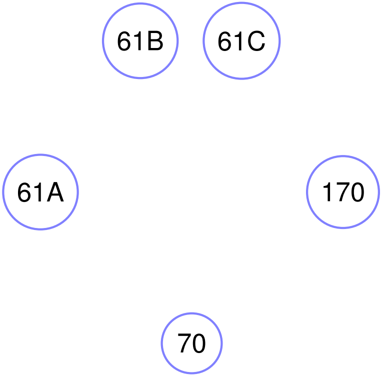




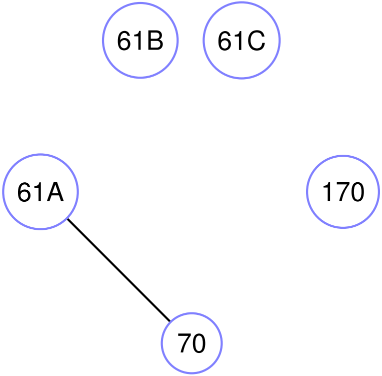
Four colors required!

Theorem: Four colors enough.

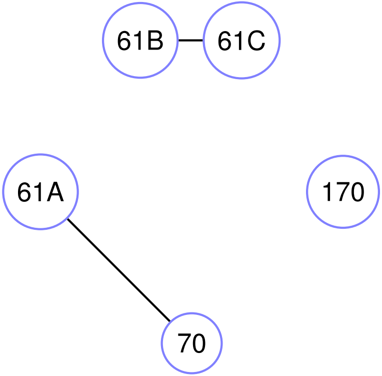
# Scheduling: coloring.



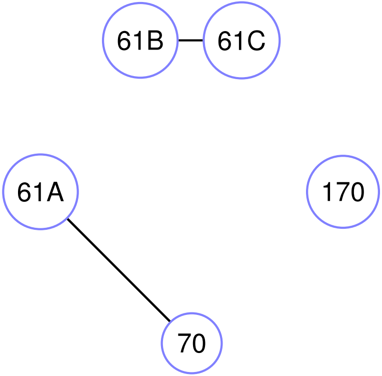
# Scheduling: coloring.



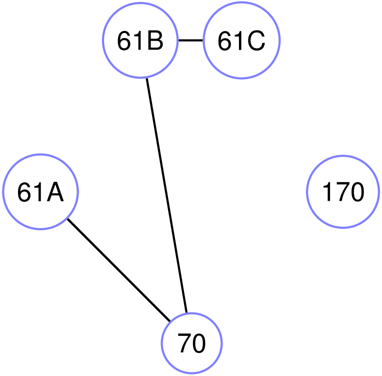
# Scheduling: coloring.



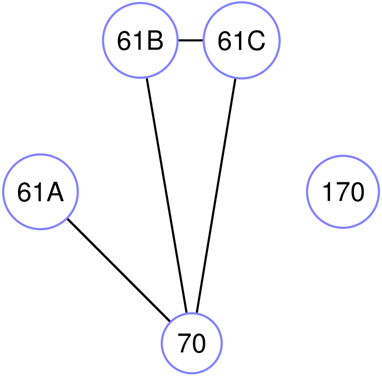
# Scheduling: coloring.



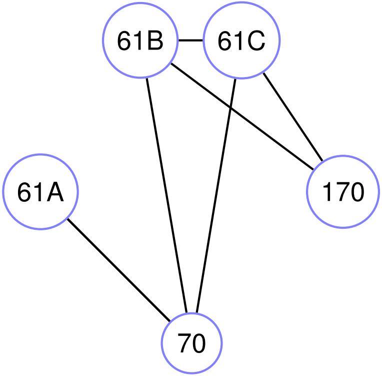
# Scheduling: coloring.



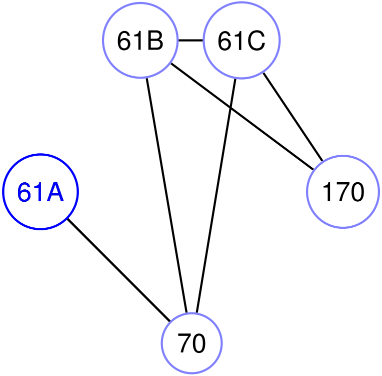
# Scheduling: coloring.



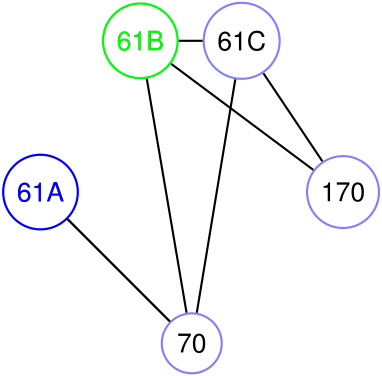
# Scheduling: coloring.



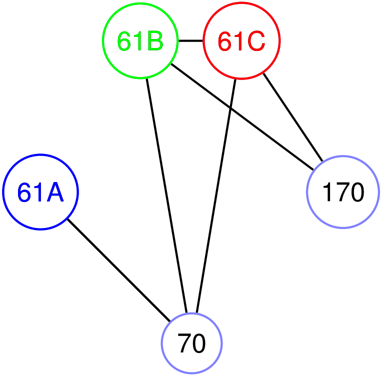
# Scheduling: coloring.



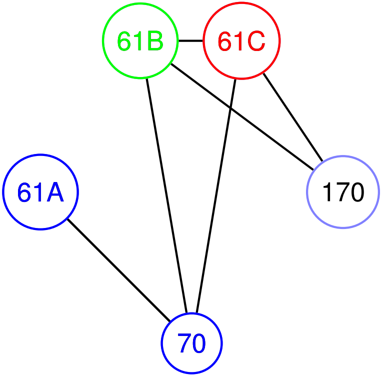
# Scheduling: coloring.



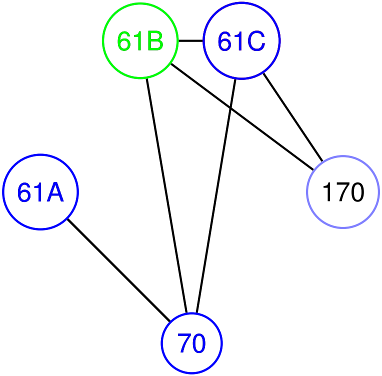
# Scheduling: coloring.



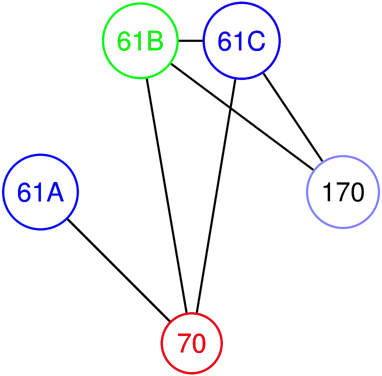
# Scheduling: coloring.



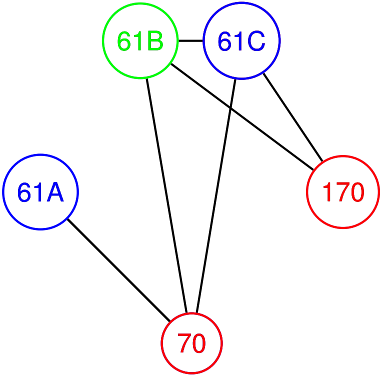
# Scheduling: coloring.



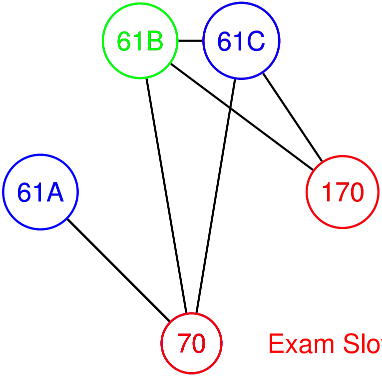
# Scheduling: coloring.



# Scheduling: coloring.



# Scheduling: coloring.



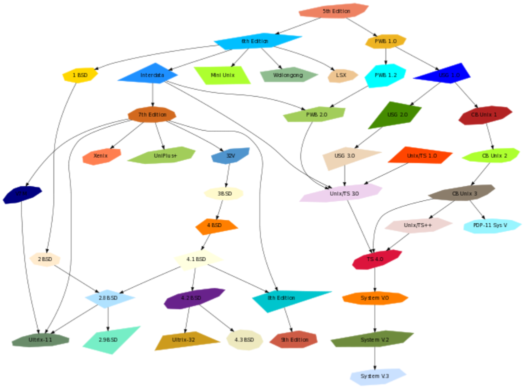

Exam Slot 1.

Exam Slot 2.

Exam Slot 3.

# Directed acyclic graphs.

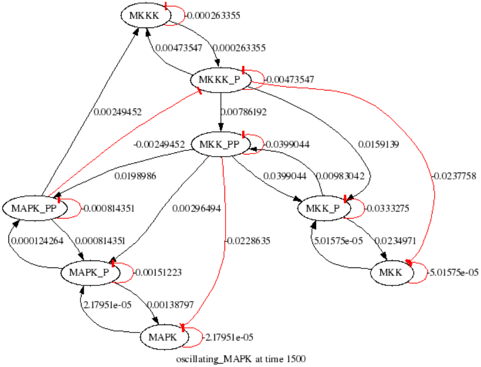
Heritage of Unix.




*Object Oriented Graphs*
*Stephen North, 3/19/93*

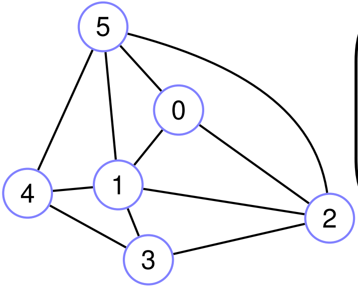
From http://www.graphviz.org/content/crazy.

# Chemical networks.




From http://www.tbi.univie.ac.at/ raim/odeSolver/doc/app.html.

Graph Implementations.



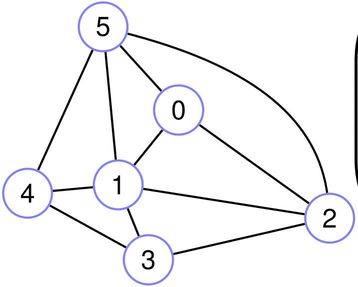

Matrix Representation.

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 \end{pmatrix}$$

$$V = \{0, 1, 2, 3, 4, 5\}$$

$$E = \{(0, 1), (0, 2), (0, 5), (1, 3) \ldots\}$$

Graph Implementations.




Matrix Representation.

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 \end{pmatrix}$$

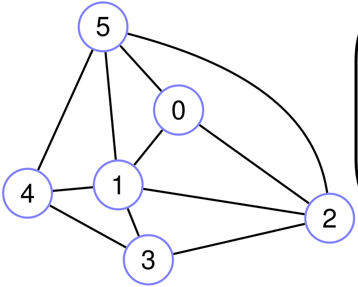
$$V = \{0, 1, 2, 3, 4, 5\}$$

$$E = \{(0, 1), (0, 2), (0, 5), (1, 3) \dots\}$$

Adjacency List

$$\begin{array}{ll} 0: & 1, 2, 5 \\ 1: & 0, 2, 3, 4, 5 \\ 2: & 0, 1, 3 \\ 3: & 1, 2, 4 \\ 4: & 1, 3, 5 \\ 5: & 0, 1, 2, 4 \end{array}$$

Graph Implementations.




Matrix Representation.

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 \end{pmatrix}$$

$$V = \{0, 1, 2, 3, 4, 5\}$$

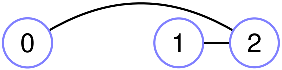
$$E = \{(0,1), (0,2), (0,5), (1,3) \ldots\}$$

Adjacency List

$0 : \quad 1, 2, 5$
$1 : \quad 0, 2, 3, 4, 5$
$2 : \quad 0, 1, 3$
$3 : \quad 1, 2, 4$
$4 : \quad 1, 3, 5$
$5 : \quad 0, 1, 2, 4$

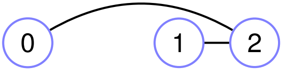
| | Matrix | Adj. List |
|---|---|---|
| Edge $(u, v)$? | $O(1)$ | $O(\|V\|)$ |
| Neighbors of $u$ | $O(\|V\|)$ | $O(d)$ |
| Space | $O(\|V\|^2)$ | $O(\|E\|)$ |

# Test your understanding..




Adjacency list of node 0?

# Test your understanding..



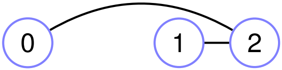

Adjacency list of node 0?

(A) $0 : 1$

(B) $0 : 1, 2$

(C) $0 : 2$

# Test your understanding..




Adjacency list of node 0?

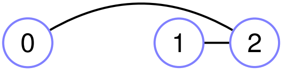
(A) $0 : 1$

(B) $0 : 1, 2$

(C) $0 : 2$

(C)

# Test your understanding..




Adjacency list of node 0?

(A) $0 : 1$

(B) $0 : 1, 2$

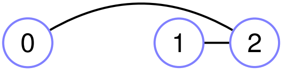
(C) $0 : 2$

(C)

How many edges?

(A) 2

# Test your understanding..




Adjacency list of node 0?

(A) $0 : 1$

(B) $0 : 1, 2$

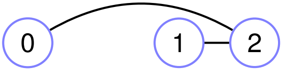
(C) $0 : 2$

(C)
How many edges?

(A) 2

Total length of adacency lists?

# Test your understanding..




Adjacency list of node 0?

(A) $0 : 1$

(B) $0 : 1, 2$

(C) $0 : 2$

(C)
How many edges?

(A) 2

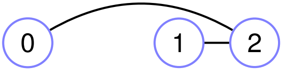
Total length of adacency lists?

(A) 2

(B) 3

(C) 4

# Test your understanding..




Adjacency list of node 0?

(A) $0 : 1$

(B) $0 : 1, 2$

(C) $0 : 2$

(C)

How many edges?

(A) 2

Total length of adacency lists?

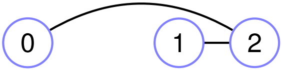
(A) 2

(B) 3

(C) 4

(C)

# Test your understanding..




Adjacency list of node 0?

(A) $0 : 1$

(B) $0 : 1, 2$

(C) $0 : 2$

(C)

How many edges?

(A) 2

Total length of adacency lists?

(A) 2

(B) 3

(C) 4

(C) 2 entries for each edge!

# Exploring a maze.

Theseus: ...

# Exploring a maze.

Theseus: ...gotta kill the minatour

# Exploring a maze.

Theseus: ...gotta kill the minatour ..in the maze

# Exploring a maze.

Theseus: ...gotta kill the minatour ..in the maze
Ariadne: he's cute..

# Exploring a maze.

Theseus: ...gotta kill the minatour ..in the maze
Ariadne: he’s cute..fortunately

# Exploring a maze.

Theseus: ...gotta kill the minatour ..in the maze
Ariadne: he's cute..fortunately ..she's smart.

# Exploring a maze.

Theseus: ...gotta kill the minatour ..in the maze
Ariadne: he's cute..fortunately ..she's smart.

Gives Theseus Ball of Thread and Chalk!

# Exploring a maze.

Theseus: ...gotta kill the minatour ..in the maze
Ariadne: he's cute..fortunately ..she's smart.

Gives Theseus Ball of Thread and Chalk!

Explore a room:

# Exploring a maze.

Theseus: ...gotta kill the minatour ..in the maze
Ariadne: he's cute..fortunately ..she's smart.

Gives Theseus Ball of Thread and Chalk!

Explore a room:
Mark room with chalk.

# Exploring a maze.

Theseus: ...gotta kill the minatour ..in the maze
Ariadne: he’s cute..fortunately ..she’s smart.

Gives Theseus Ball of Thread and Chalk!

Explore a room:
Mark room with chalk.
For each exit.

# Exploring a maze.

Theseus: ...gotta kill the minatour ..in the maze
Ariadne: he's cute..fortunately ..she's smart.

Gives Theseus Ball of Thread and Chalk!

Explore a room:
Mark room with chalk.
For each exit.
  Look through exit. If marked, next exit.

# Exploring a maze.

Theseus: ...gotta kill the minatour ..in the maze
Ariadne: he's cute..fortunately ..she's smart.

Gives Theseus Ball of Thread and Chalk!

Explore a room:
Mark room with chalk.
For each exit.
  Look through exit. If marked, next exit.
  Otherwise go in room unwind thread.

# Exploring a maze.

Theseus: ...gotta kill the minatour ..in the maze
Ariadne: he's cute..fortunately ..she's smart.

Gives Theseus Ball of Thread and Chalk!

Explore a room:
Mark room with chalk.
For each exit.
  Look through exit. If marked, next exit.
  Otherwise go in room unwind thread.
    Explore that room.

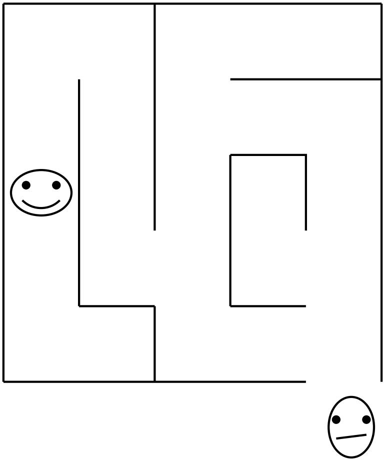
# Exploring a maze.

Theseus: ...gotta kill the minatour ..in the maze
Ariadne: he's cute..fortunately ..she's smart.

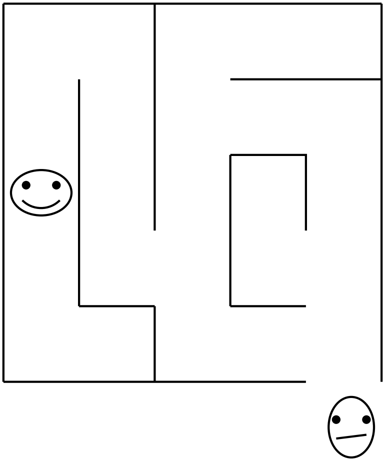
Gives Theseus Ball of Thread and Chalk!

Explore a room:
Mark room with chalk.
For each exit.
    Look through exit. If marked, next exit.
    Otherwise go in room unwind thread.
        Explore that room.
Wind thread to go back to "previous" room.

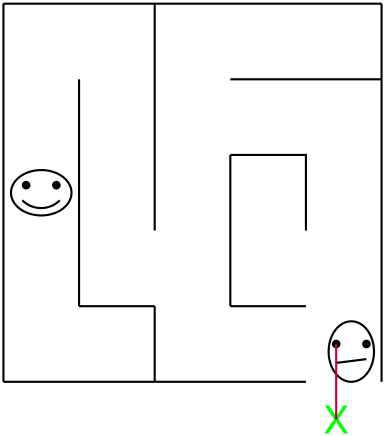
Where is the minatour?



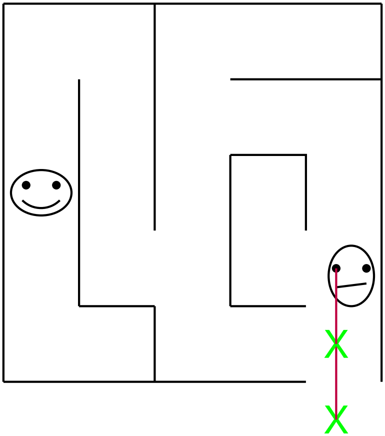
Where is the minatour?



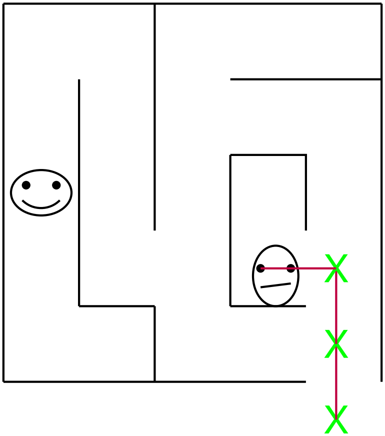
Where is the minatour?



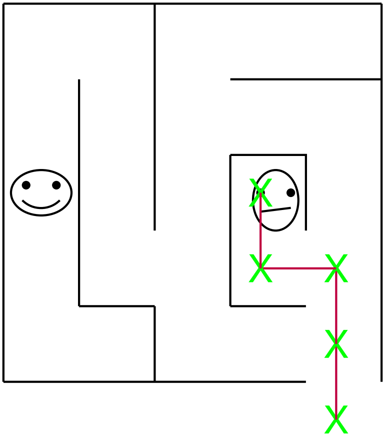
Where is the minatour?



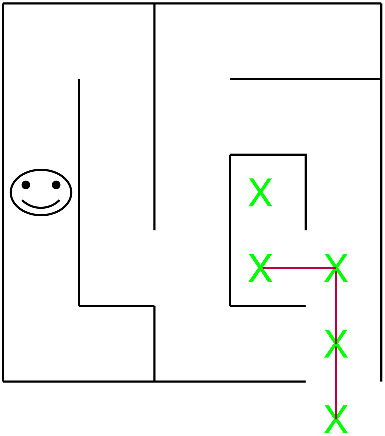
Where is the minatour?



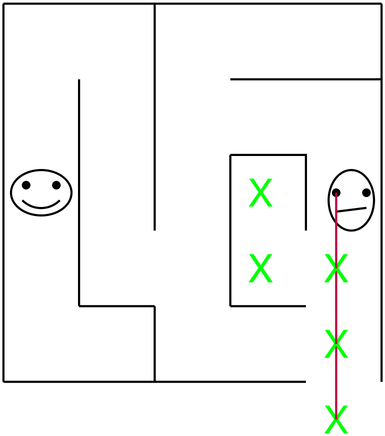
Where is the minatour?



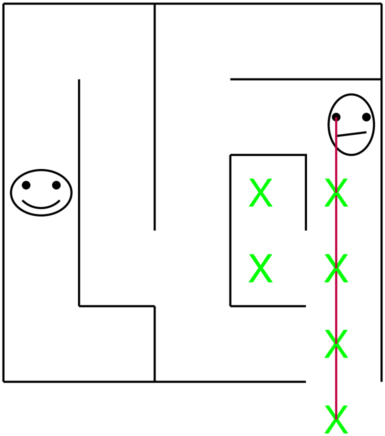
Where is the minatour?



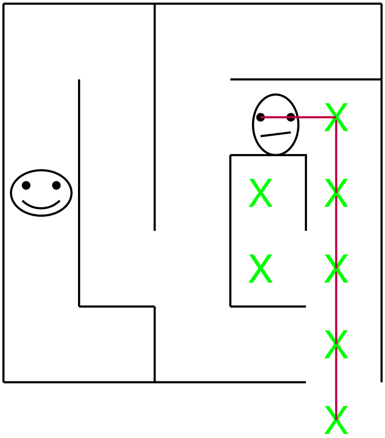
Where is the minatour?



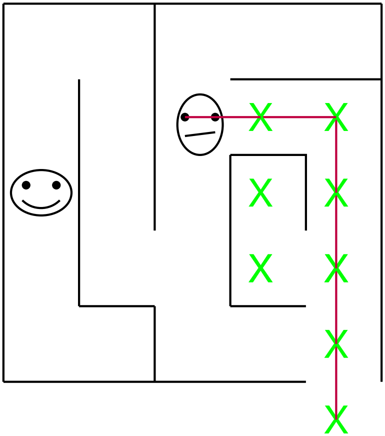
Where is the minatour?



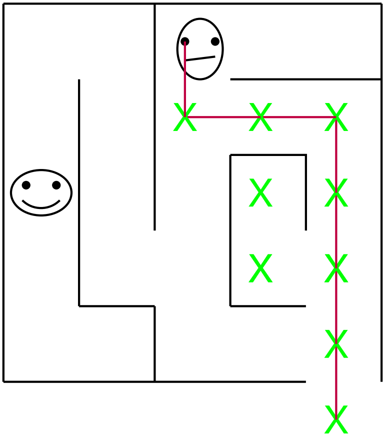
Where is the minatour?



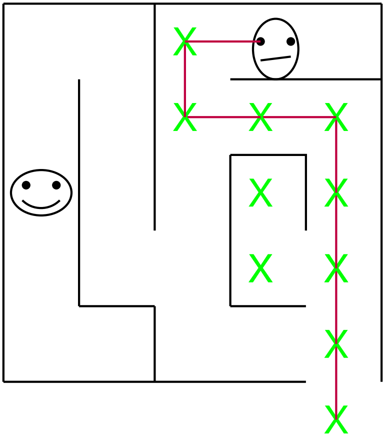
Where is the minatour?



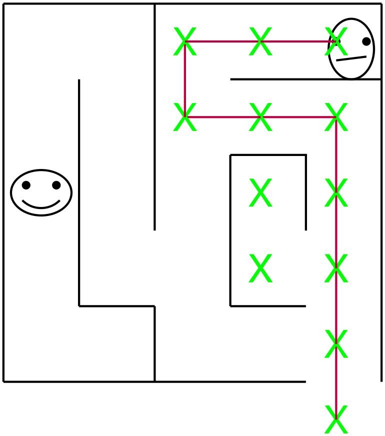
Where is the minatour?



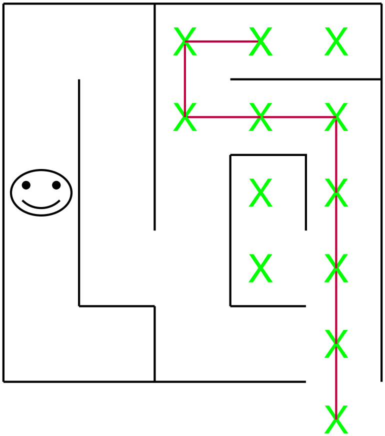
Where is the minatour?



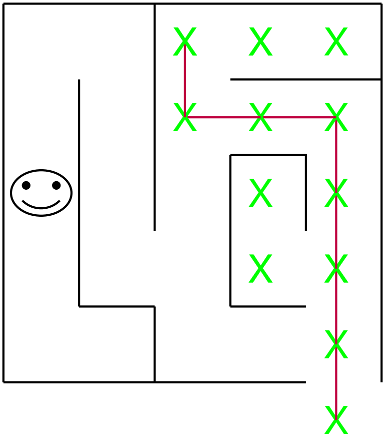
Where is the minatour?



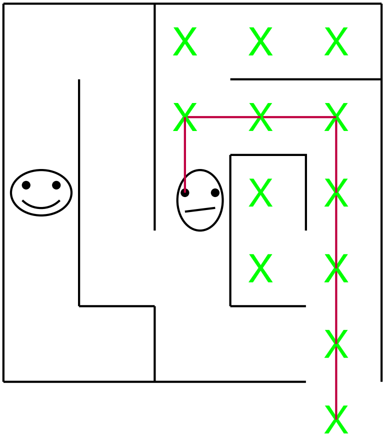
Where is the minatour?



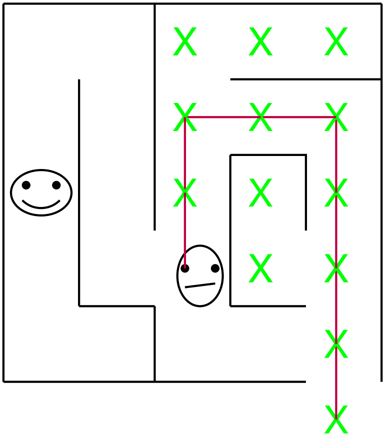
Where is the minatour?



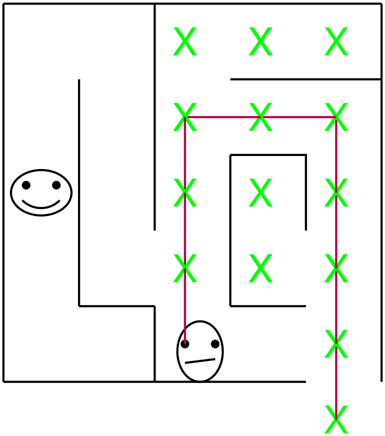
Where is the minatour?



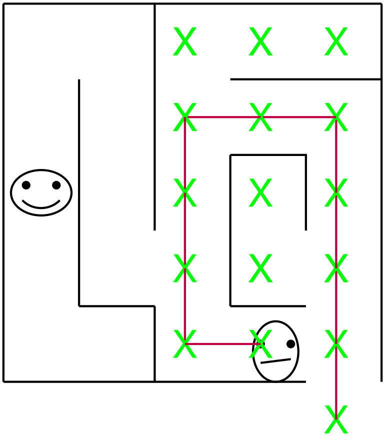
Where is the minatour?



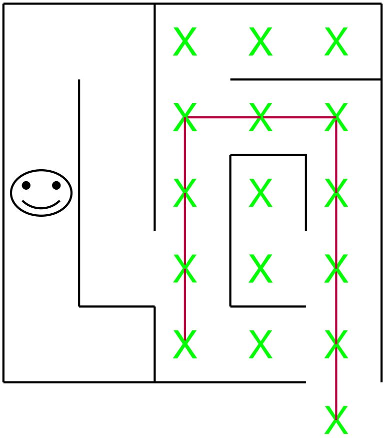
Where is the minatour?



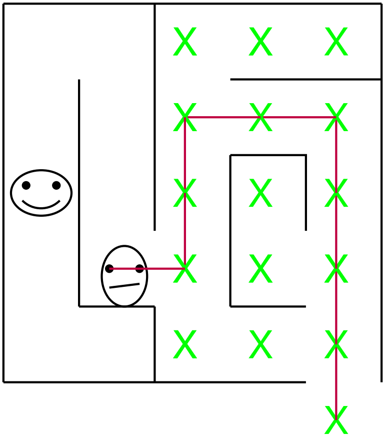
Where is the minatour?



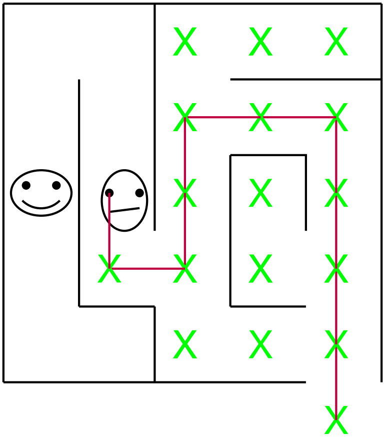
Where is the minatour?



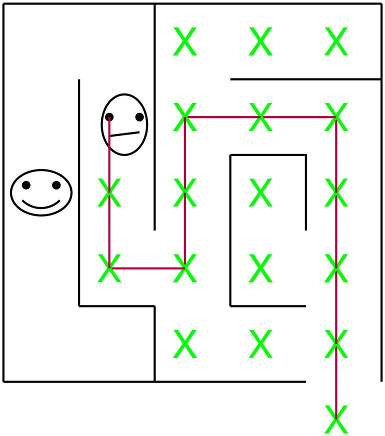
Where is the minatour?



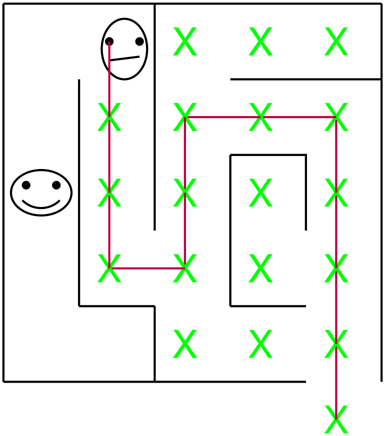
Where is the minatour?



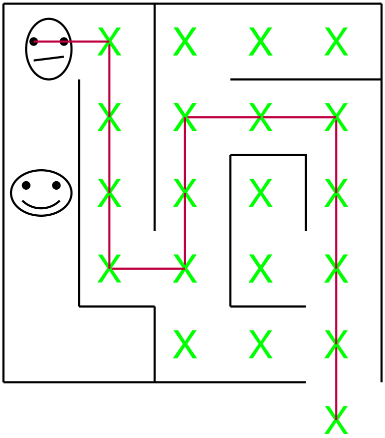
Where is the minatour?



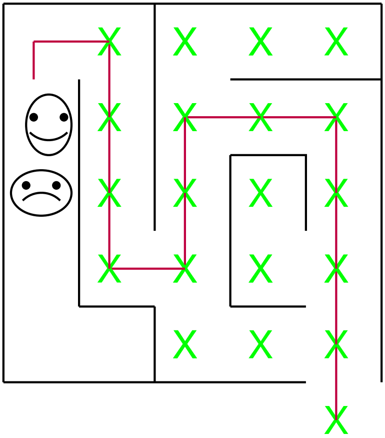
Where is the minatour?



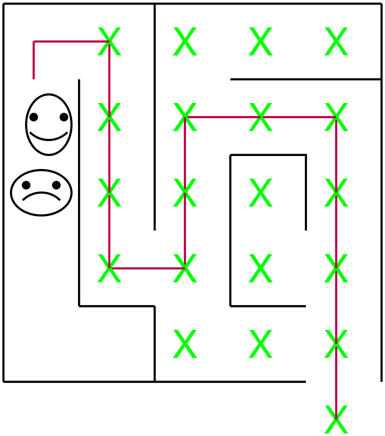
Where is the minatour?



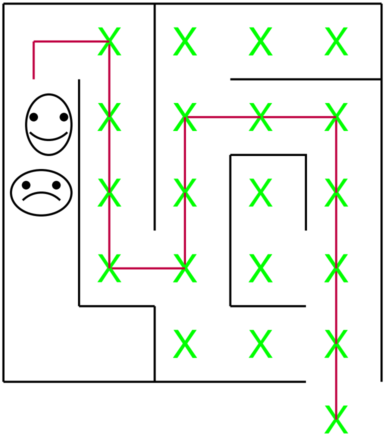
Where is the minatour?



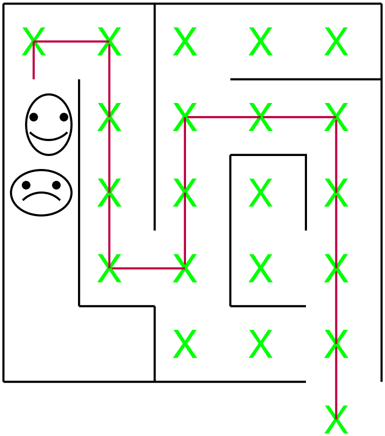
Where is the minatour?



Where is the minatour?



Where is the minatour?



# Searching

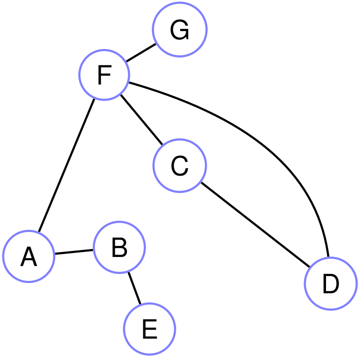
Find a minatour!

# Searching

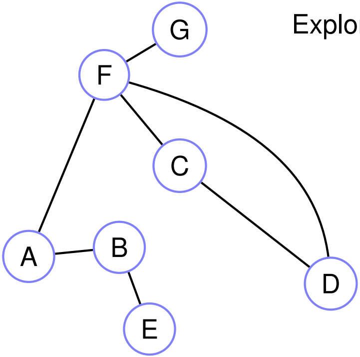
Find a minatour!

Find out which nodes are reachable from $A$.

# Explore.



# Explore.

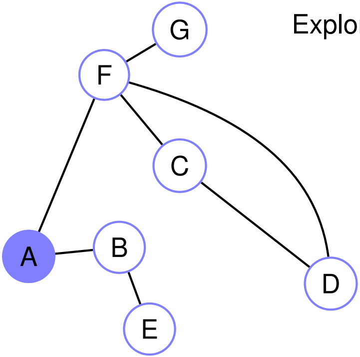



Explore(v):

1. Set visited[v] := true
2. for each edge (v,w) in E
3. if not visited[w]: Explore(w).

Chalk.
Stack is Thread.

# Explore.

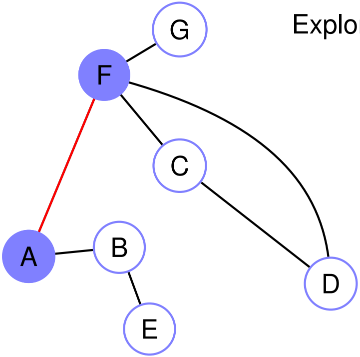



Explore(v):

1. Set visited[v] := true
2. for each edge (v,w) in E
3. if not visited[w]: Explore(w).

Chalk.
Stack is Thread.

# Explore.



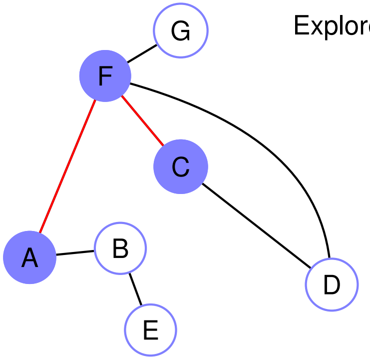

Explore(v):

1. Set visited[v] := true
2. for each edge (v,w) in E
3. if not visited[w]: Explore(w).

Chalk.

Stack is Thread.

# Explore.



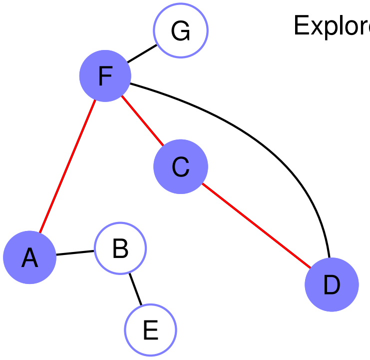

Explore(v):

1. Set visited[v] := true
2. for each edge (v,w) in E
3. if not visited[w]: Explore(w).

Chalk.

Stack is Thread.

# Explore.



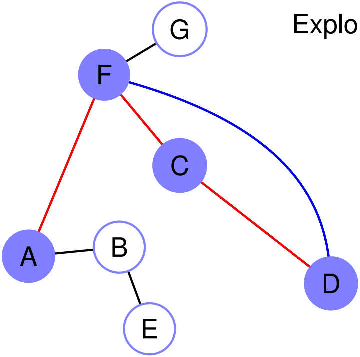

Explore(v):

1. Set visited[v] := true
2. for each edge (v,w) in E
3. if not visited[w]: Explore(w).

Chalk.

Stack is Thread.

# Explore.



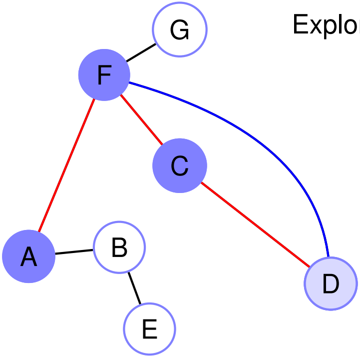

Explore(v):

1. Set visited[v] := true
2. for each edge (v,w) in E
3. if not visited[w]: Explore(w).

Chalk.

Stack is Thread.

# Explore.



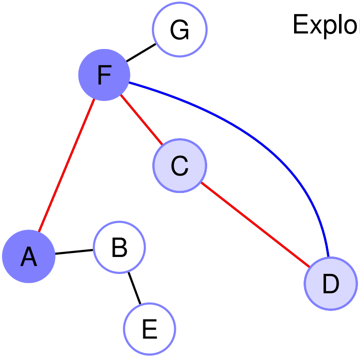

Explore(v):

1. Set visited[v] := true
2. for each edge (v,w) in E
3. if not visited[w]: Explore(w).

Chalk.

Stack is Thread.

# Explore.

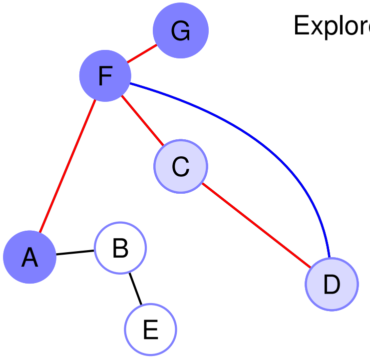



Explore(v):

1. Set visited[v] := true
2. for each edge (v,w) in E
3. if not visited[w]: Explore(w).

Chalk.
Stack is Thread.

# Explore.



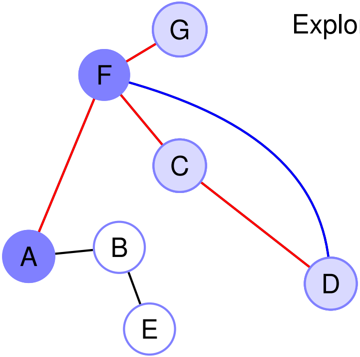

Explore(v):

1. Set visited[v] := true
2. for each edge (v,w) in E
3. if not visited[w]: Explore(w).

Chalk.

Stack is Thread.

# Explore.

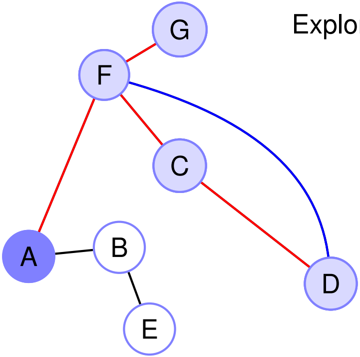



Explore(v):

1. Set visited[v] := true
2. for each edge (v,w) in E
3. if not visited[w]: Explore(w).

Chalk.
Stack is Thread.

# Explore.



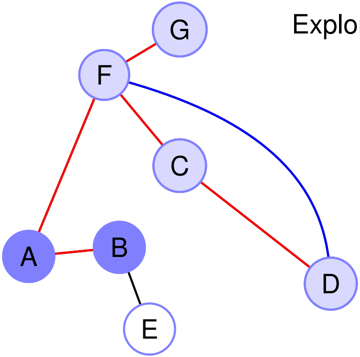

Explore(v):

1. Set visited[v] := true
2. for each edge (v,w) in E
3. if not visited[w]: Explore(w).

Chalk.

Stack is Thread.

# Explore.

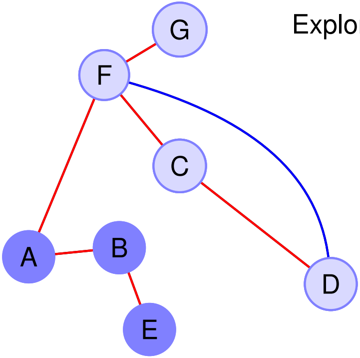



Explore(v):

1. Set visited[v] := true
2. for each edge (v,w) in E
3. if not visited[w]: Explore(w).

Chalk.
Stack is Thread.

# Explore.



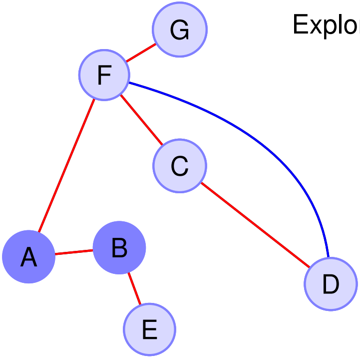

Explore(v):

1. Set visited[v] := true
2. for each edge (v,w) in E
3. if not visited[w]: Explore(w).

Chalk.

Stack is Thread.

# Explore.



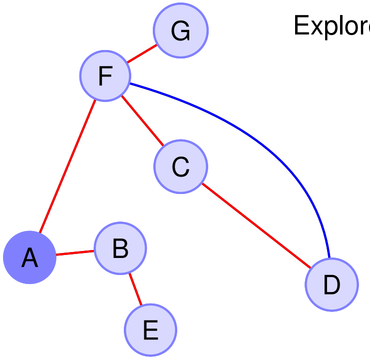

Explore(v):

1. Set visited[v] := true
2. for each edge (v,w) in E
3. if not visited[w]: Explore(w).

Chalk.

Stack is Thread.

# Explore.



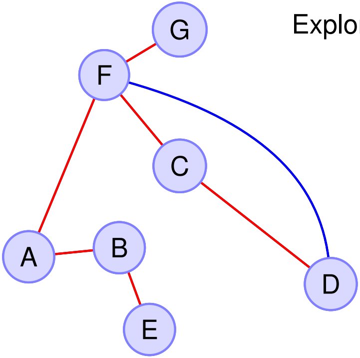

Explore(v):

1. Set visited[v] := true
2. for each edge (v,w) in E
3. if not visited[w]: Explore(w).

Chalk.

Stack is Thread.

# Explore.



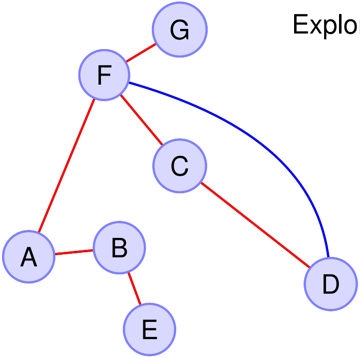

Explore(v):

1. Set visited[v] := true
2. for each edge (v,w) in E
3. if not visited[w]: Explore(w).

Chalk.

Stack is Thread.

# Explore.




Explore(v):

1. Set visited[v] := true
2. for each edge (v,w) in E
3. if not visited[w]: Explore(w).

Chalk.
Stack is Thread.

Explore builds tree.

*Tree* and *back* edges.

# Correctness.

**Explore(v):**

1. Set visited[v] := **true**.
2. For each edge (v,w) in E
3.     if not visited[w]: Explore(w)

# Correctness.

**Explore(v):**

1. Set visited[v] := **true**.
2. For each edge (v,w) in E
3.     if not visited[w]: Explore(w)

**Property:**

All and only nodes reachable from $A$ are reached by explore.

# Correctness.

**Explore(v):**

1. Set visited[v] := **true**.
2. For each edge (v,w) in E
3. if not visited[w]: Explore(w)

**Property:**

All and only nodes reachable from $A$ are reached by explore.

Only: when $u$ visited.

# Correctness.

**Explore(v):**

1. Set visited[v] := **true**.
2. For each edge (v,w) in E
3. if not visited[w]: Explore(w)

**Property:**

All and only nodes reachable from $A$ are reached by explore.

Only: when $u$ visited.

stack contains nodes in a path from $a$ to $u$.

# Correctness.

**Explore(v):**
1. Set visited[v] := **true**.
2. For each edge (v,w) in E
3.     if not visited[w]: Explore(w)

**Property:**
All and only nodes reachable from $A$ are reached by explore.

Only: when $u$ visited.
  stack contains nodes in a path from $a$ to $u$.

All: if a node $u$ is reachable.
  there is a path to it. Assume: $u$ not found.

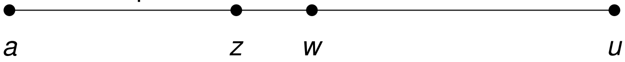
# Correctness.

**Explore(v):**
1. Set visited[v] := **true**.
2. For each edge (v,w) in E
3.     if not visited[w]: Explore(w)

**Property:**
All and only nodes reachable from *A* are reached by explore.

Only: when *u* visited.
  stack contains nodes in a path from *a* to *u*.

All: if a node *u* is reachable.
  there is a path to it. Assume: *u* not found.



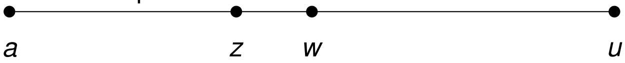
# Correctness.

**Explore(v):**
1. Set visited[v] := **true**.
2. For each edge (v,w) in E
3.     if not visited[w]: Explore(w)

**Property:**
All and only nodes reachable from *A* are reached by explore.

Only: when *u* visited.
stack contains nodes in a path from *a* to *u*.

All: if a node *u* is reachable.
there is a path to it. Assume: *u* not found.




*z* is explored.

# Correctness.

**Explore(v):**

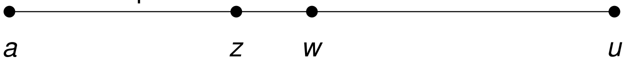
1. Set visited[v] := **true**.
2. For each edge (v,w) in E
3.     if not visited[w]: Explore(w)

**Property:**

All and only nodes reachable from *A* are reached by explore.

Only: when *u* visited.
  stack contains nodes in a path from *a* to *u*.

All: if a node *u* is reachable.
  there is a path to it. Assume: *u* not found.




*z* is explored. *w* is not!

# Correctness.

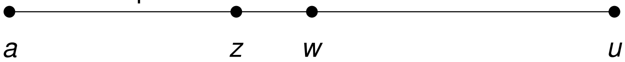
**Explore(v):**

```
Set visited[v] := true.
For each edge (v,w) in E
    if not visited[w]: Explore(w)
```

**Property:**
All and only nodes reachable from *A* are reached by explore.

Only: when *u* visited.
stack contains nodes in a path from *a* to *u*.

All: if a node *u* is reachable.
there is a path to it. Assume: *u* not found.




*z* is explored. *w* is not!
Explore (*z*) would explore(*w*)!

# Correctness.

**Explore(v):**

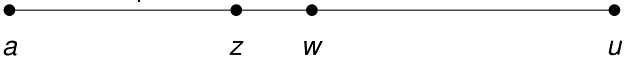
1. Set visited[v] := **true**.
2. For each edge (v,w) in E
3.   if not visited[w]: Explore(w)

**Property:**

All and only nodes reachable from $A$ are reached by explore.

Only: when $u$ visited.
  stack contains nodes in a path from $a$ to $u$.

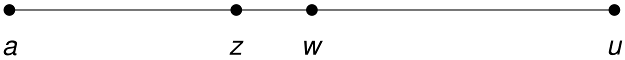
All: if a node $u$ is reachable.
  there is a path to it. Assume: $u$ not found.




$z$ is explored. $w$ is not!
Explore ($z$) would explore($w$)! Contradiction.

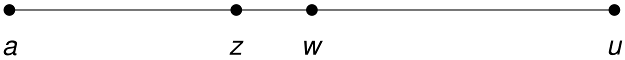
$\square$

# Proof was induction.




Property: Every node with a path of length $k$ or less is reached.

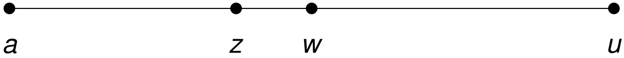
# Proof was induction.




Property: Every node with a path of length $k$ or less is reached.

Induction by Contradiction.

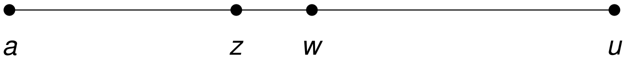
# Proof was induction.




Property: Every node with a path of length $k$ or less is reached.

Induction by Contradiction.
Find smallest $k$ (path length) where property doesn't hold.

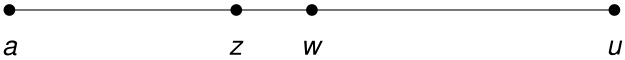
# Proof was induction.




Property: Every node with a path of length $k$ or less is reached.

Induction by Contradiction.
Find smallest $k$ (path length) where property doesn't hold.
It does hold.

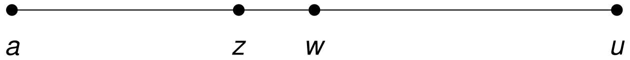
# Proof was induction.




Property: Every node with a path of length $k$ or less is reached.

Induction by Contradiction.
Find smallest $k$ (path length) where property doesn't hold.
It does hold.
No smallest $k$ where it fails.

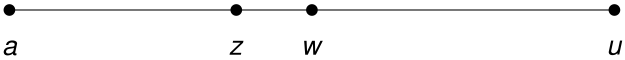
# Proof was induction.




Property: Every node with a path of length $k$ or less is reached.

Induction by Contradiction.
Find smallest $k$ (path length) where property doesn't hold.
It does hold.
No smallest $k$ where it fails.
Must hold for every $k$.

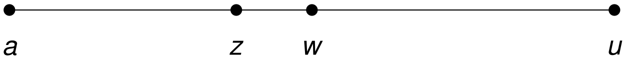
# Proof was induction.




Property: Every node with a path of length *k* or less is reached.

Induction by Contradiction.
Find smallest *k* (path length) where property doesn't hold.
It does hold.
No smallest *k* where it fails.
Must hold for every *k*.
Done!!!

# Proof was induction.




Property: Every node with a path of length $k$ or less is reached.

Induction by Contradiction.
Find smallest $k$ (path length) where property doesn't hold.
It does hold.
No smallest $k$ where it fails.
Must hold for every $k$.
Done!!! or $\square$.

# Running Time.

**Explore(v):**

1. Set visited[v] := **true**.
2. For each edge (v,w) in E
3.     if not visited[w]: Explore(w).

# Running Time.

**Explore(v):**

1. Set visited[v] := **true**.
2. For each edge (v,w) in E
3.     if not visited[w]: Explore(w).

How to analyse?

# Running Time.

**Explore(v):**

1. Set visited[v] := **true**.
2. For each edge (v,w) in E
3.     if not visited[w]: Explore(w).

How to analyse?

Let $n = |V|$, and $m = |E|$.

# Running Time.

**Explore(v):**

1. Set visited[v] := **true**.
2. For each edge (v,w) in E
3. if not visited[w]: Explore(w).

How to analyse?

Let $n = |V|$, and $m = |E|$.

$T(n,m) \leq (d)T(n-1,m) + O(d)$

# Running Time.

**Explore(v):**

1. Set visited[v] := **true**.
2. For each edge (v,w) in E
3. if not visited[w]: Explore(w).

How to analyse?

Let $n = |V|$, and $m = |E|$.

$T(n,m) \leq (d)T(n-1,m) + O(d)$ Exponential ?!?!?!

# Running Time.

**Explore(v):**

1. Set visited[v] := **true**.
2. For each edge (v,w) in E
3.   if not visited[w]: Explore(w).

How to analyse?

Let $n = |V|$, and $m = |E|$.

$T(n,m) \leq (d)T(n-1,m) + O(d)$  Exponential ?!?!?!

Don't use recurrence!

# Running Time.

**Explore(v):**

1. Set visited[v] := **true**.
2. For each edge (v,w) in E
3.     if not visited[w]: Explore(w).

# Running Time.

**Explore(v):**

1. Set visited[v] := **true**.
2. For each edge (v,w) in E
3.     if not visited[w]: Explore(w).

How to analyse?

# Running Time.

**Explore(v):**

1. Set visited[v] := **true**.
2. For each edge (v,w) in E
3.     if not visited[w]: Explore(w).

How to analyse?

Let $n = |V|$, and $m = |E|$.

# Running Time.

**Explore(v):**

1. Set visited[v] := **true**.
2. For each edge (v,w) in E
3.     if not visited[w]: Explore(w).

How to analyse?

Let $n = |V|$, and $m = |E|$.

"Charge work to something."

# Running Time.

**Explore(v):**

1. Set visited[v] := **true**.
2. For each edge (v,w) in E
3.     if not visited[w]: Explore(w).

How to analyse?

Let $n = |V|$, and $m = |E|$.

“Charge work to something.”

For node $x$:

# Running Time.

**Explore(v):**

1. Set visited[v] := **true**.
2. For each edge (v,w) in E
3.     if not visited[w]: Explore(w).

How to analyse?

Let $n = |V|$, and $m = |E|$.

"Charge work to something."

For node $x$:
  Explore once!

# Running Time.

**Explore(v):**

1. Set visited[v] := **true**.
2. For each edge (v,w) in E
3.     if not visited[w]: Explore(w).

How to analyse?

Let $n = |V|$, and $m = |E|$.

"Charge work to something."

For node $x$:

Explore once!
Process each incident edge.

# Running Time.

**Explore(v):**

1. Set visited[v] := **true**.
2. For each edge (v,w) in E
3.     if not visited[w]: Explore(w).

How to analyse?

Let $n = |V|$, and $m = |E|$.

"Charge work to something."

For node $x$:

Explore once!

Process each incident edge.

Each edge processed twice.

# Running Time.

**Explore(v):**

1. Set visited[v] := **true**.
2. For each edge (v,w) in E
3.   if not visited[w]: Explore(w).

How to analyse?

Let $n = |V|$, and $m = |E|$.

"Charge work to something."

For node $x$:
  Explore once!
  Process each incident edge.

Each edge processed twice.

$O(n)$ - call explore on $n$ nodes.

# Running Time.

**Explore(v):**

1. Set visited[v] := **true**.
2. For each edge (v,w) in E
3. if not visited[w]: Explore(w).

How to analyse?

Let $n = |V|$, and $m = |E|$.

"Charge work to something."

For node $x$:

Explore once!
Process each incident edge.

Each edge processed twice.

$O(n)$ - call explore on $n$ nodes.
$O(m)$ - process each edge twice.

# Running Time.

**Explore(v):**
1. Set visited[v] := **true**.
2. For each edge (v,w) in E
3. if not visited[w]: Explore(w).

How to analyse?

Let $n = |V|$, and $m = |E|$.

"Charge work to something."

For node $x$:
 Explore once!
 Process each incident edge.

Each edge processed twice.

$O(n)$ - call explore on $n$ nodes.
$O(m)$ - process each edge twice.
Total: $O(n+m)$.

# Lecture in a minute.

FFT Wrapup:

# Lecture in a minute.

FFT Wrapup:

- Evaluate degree $n$ polynomial on $n$ points.
- Recursion:
  - Evaluate Odd/Even polynomials on squares of points.
- Which $n = 2^k$ points?

# Lecture in a minute.

FFT Wrapup:
Evaluate degree $n$ polynomial on $n$ points.
Recursion:
Evaluate Odd/Even polynomials on squares of points.
Which $n = 2^k$ points?
$n$th root of unity: complex numbers.
"squares of $n$th root of unity are $n/2$th roots of unity."
Evaluate Odd/Even polynomials on $n/2$ points. $\Longrightarrow$
$T(n) = 2T(n/2) + O(n) = O(n \log n)$.

# Lecture in a minute.

FFT Wrapup:

Evaluate degree $n$ polynomial on $n$ points.

Recursion:

Evaluate Odd/Even polynomials on squares of points.

Which $n = 2^k$ points?

$n$th root of unity: complex numbers.

"squares of $n$th root of unity are $n/2$th roots of unity."

Evaluate Odd/Even polynomials on $n/2$ points. $\implies$

$T(n) = 2T(n/2) + O(n) = O(n \log n)$.

Graphs:

# Lecture in a minute.

FFT Wrapup:

Evaluate degree $n$ polynomial on $n$ points.

Recursion:

Evaluate Odd/Even polynomials on squares of points.

Which $n = 2^k$ points?

$n$th root of unity: complex numbers.

"squares of $n$th root of unity are $n/2$th roots of unity."

Evaluate Odd/Even polynomials on $n/2$ points. $\implies$

$T(n) = 2T(n/2) + O(n) = O(n \log n)$.

Graphs:

$G = (V,E)$, $V$ - vertices, $E$ edges.

Representations:

Matrix: $|V|^2$ space, fast check for edges.

Adjacency List: $O(|V| + |E|)$ space. More complicated.

# Lecture in a minute.

FFT Wrapup:

Evaluate degree $n$ polynomial on $n$ points.

Recursion:

Evaluate Odd/Even polynomials on squares of points.

Which $n = 2^k$ points?

$n$th root of unity: complex numbers.

"squares of $n$th root of unity are $n/2$th roots of unity."

Evaluate Odd/Even polynomials on $n/2$ points. $\implies$

$T(n) = 2T(n/2) + O(n) = O(n \log n)$.

Graphs:

$G = (V, E)$, $V$ - vertices, $E$ edges.

Representations:

Matrix: $|V|^2$ space, fast check for edges.

Adjacency List: $O(|V| + |E|)$ space. More complicated.

Procedure: explore($v$).

Explores the graph.

Uses a stack.

Nonrecursive non-loop counting Runtime analysis.

Every little move she makes...