CSE 421

Dynamic Programming

Shortest Paths with Negative Weights

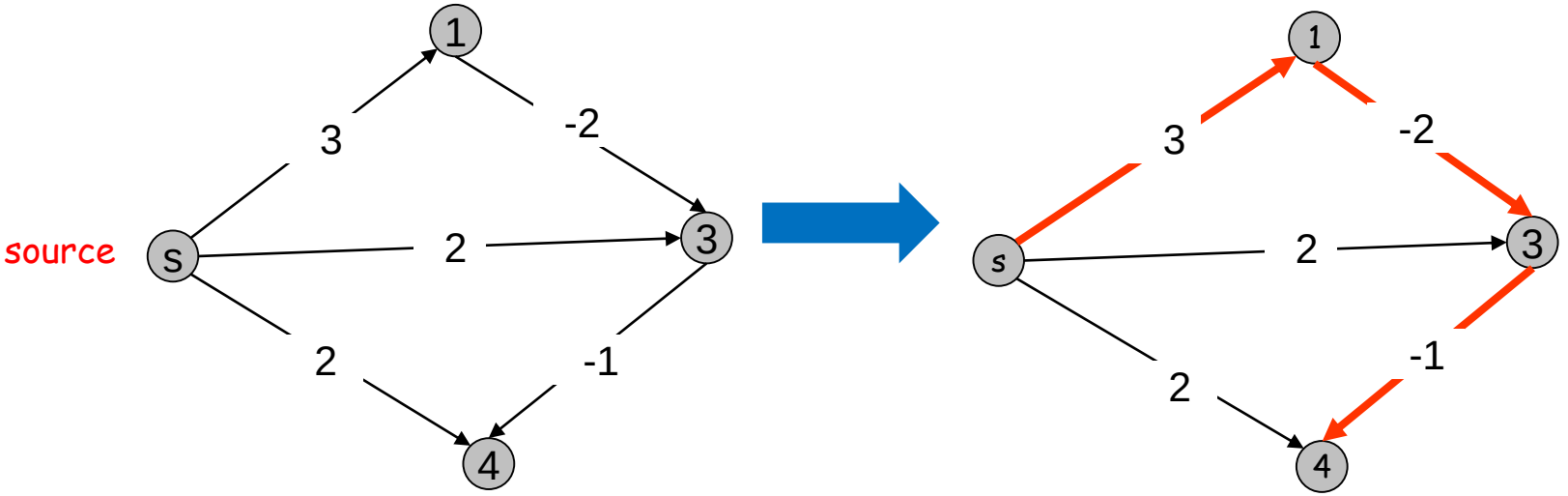
Yin Tat Lee

Shortest Paths with Neg Edge Weights

Given a weighted directed graph $G = (V, E)$ and a source vertex s, where the weight of edge (u,v) is $c_{u,v}$ **(that can be negative)**

Goal: Find the shortest path from s to all vertices of G.

Recall that Dikjstra's Algorithm fails when weights are negative

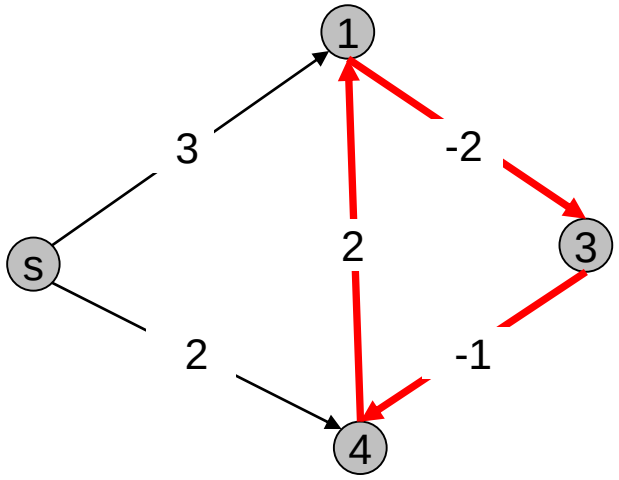


Impossibility on Graphs with Neg Cycles

Observation: No solution exists if G has a negative cycle.

This is because we can minimize the length by going over the cycle again and again.

So, suppose G does not have a negative cycle.



DP for Shortest Path (First Attempt)

Def: Let $OPT(v)$ be the length of the shortest s - v path

$$OPT(v) = \begin{cases} 0 & \text{if } v = s \\ \min_{u:(u,v) \text{ an edge}} OPT(u) + c_{u,v} \end{cases}$$

The formula is correct. But it is not clear how to compute it.

DP for Shortest Path

Def: Let $OPT(v, i)$ be the length of the shortest s - v path with at most i edges.

Let us characterize $OPT(v, i)$.

Case 1: $OPT(v, i)$ path has less than i edges.

- Then, $OPT(v, i) = OPT(v, i - 1)$.

Case 2: $OPT(v, i)$ path has exactly i edges.

- Let $s, v_1, v_2, \dots, v_{i-1}, v$ be the $OPT(v, i)$ path with i edges.
- Then, $s, v_1, \dots, v_{i-1}$ must be the shortest s - v_{i-1} path with at most $i - 1$ edges. So,

$$OPT(v, i) = OPT(v_{i-1}, i - 1) + c_{v_{i-1}, v}$$

DP for Shortest Path

Def: Let $OPT(v,i)$ be the length of the shortest s - v path with at most i edges.

$$OPT(v,i) = \begin{cases} 0 & \text{if } v = s \\ \infty & \text{if } v \neq s, i = 0 \\ \min(OPT(v,i-1), \min_{u:(u,v) \text{ an edge}} OPT(u,i-1) + c_{u,v}) & \end{cases}$$

So, for every v, $OPT(v,?)$ is the shortest path from s to v.

But how long do we have to run?

Since G has no negative cycle, it has at most $n-1$ edges. So, $OPT(v,n-1)$ is the answer.

Bellman Ford Algorithm

```
for v=1 to n
    if v ≠ s then
        M[v,0]=∞
M[s,0]=0.

for i=1 to n-1
    for v=1 to n
        M[v,i]=M[v,i-1]
        for every edge (u,v)
            M[v,i]=min(M[v,i], M[u,i-1]+c_{u,v})
```

	Complexity	Author
	$O(n^4)$	Shimbel (1955) [30]
	$O(Wn^2m)$	Ford (1956) [14]
*	$O(nm)$	Bellman (1958) [1], Moore (1959) [25]
	$O(n^{\frac{3}{4}}m \log W)$	Gabow (1983) [9]
	$O(\sqrt{nm}\log(nW))$	Gabow and Tarjan (1989) [10]
*	$O(\sqrt{nm}\log(W))$	Goldberg (1993) [12]
*	$\tilde{O}(Wn^\omega)$	Sankowski (2005) [27] Yuster and Zwick (2005) [35]
*	$\tilde{O}(m^{10/7}\log W)$	Cohen, Madry, Sankowski, Vladu (2016)

Table 1: The complexity results for the SSSP problem with negative weights (* indicates asymptotically the best bound for some range of parameters).

Running Time: $O(nm)$

Can we test if G has negative cycles?

Yes, run for i=1…3n and see if the M[v,n-1] is different from M[v,3n]

DP Techniques Summary

Recipe:

- Follow the natural induction proof.
- Find out additional assumptions/variables/subproblems that you need to do the induction
- Strengthen the hypothesis and define w.r.t. new subproblems

Dynamic programming techniques.

- Ordering is important: longest path in DAG
- Adding a new variable: knapsack.
- Dynamic programming over intervals: RNA secondary structure.

Top-down vs. bottom-up:

- Different people have different intuitions
- Bottom-up is useful to optimize the memory

CSE 421

Divide and Conquer

$O(n\ logn)$ time polynomial multiplication

Multiplication

- Polynomials

  Naïve: $\Theta(n^2)$

  Karatsuba: $\Theta(n^{1.59\ldots})$

  Best known: $\Theta(n \log n)$

  - "Fast Fourier Transform“
  - FFT widely used for signal processing

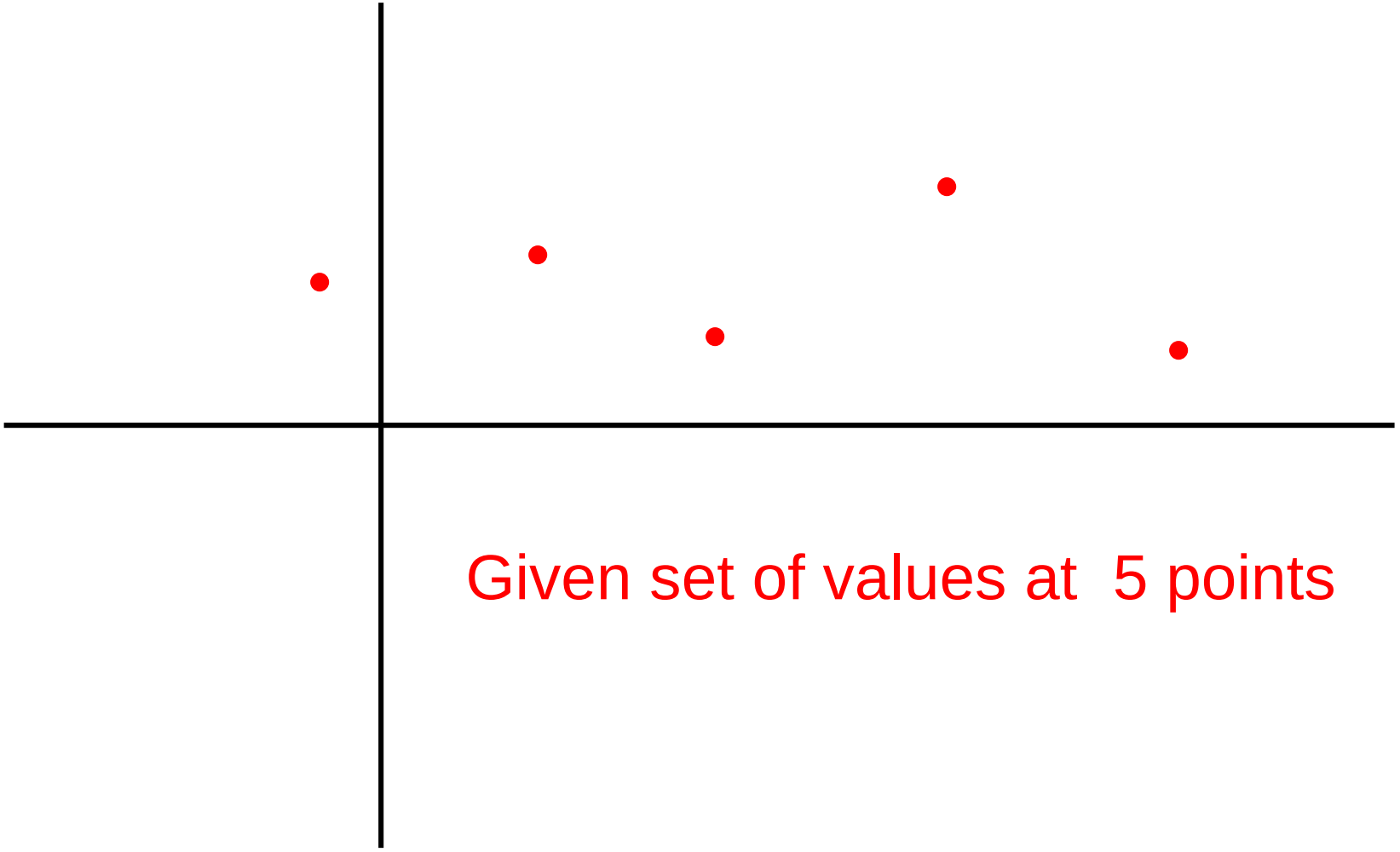
- Integers

  Similar, but some ugly details re: carries, etc. due to Schonhage-Strassen in 1971 gives $\Theta(n \log n \log\log n)$

  Improvement in 2007 due to Furer gives $\Theta(n \log n \, 2^{\log^* n})$

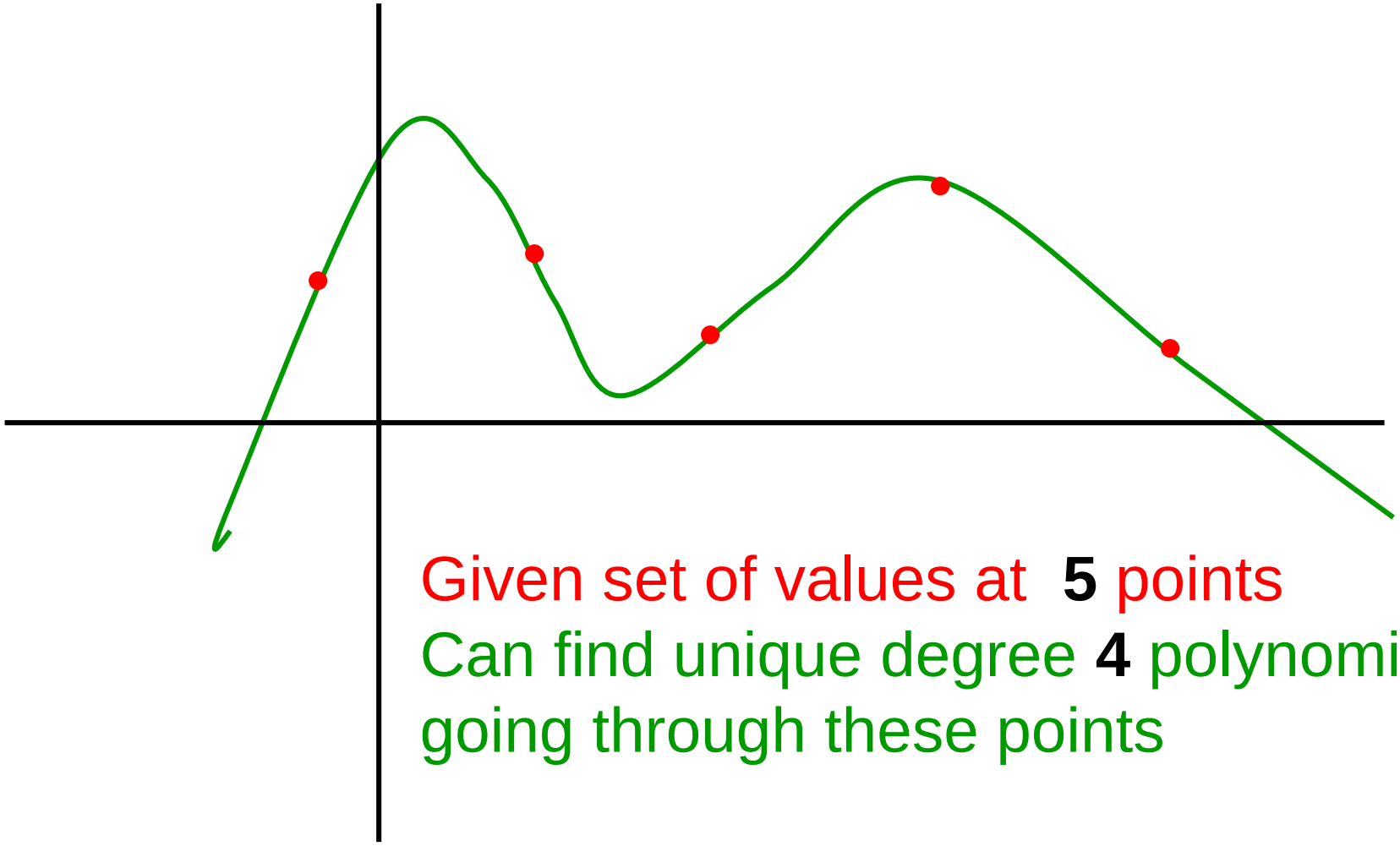
  Used in practice in symbolic manipulation systems like Maple

Interpolation



Given set of values at 5 points

Interpolation



Given set of values at **5** points
Can find unique degree **4** polynomial going through these points

Multiplying Polynomials by Evaluation & Interpolation

- Any degree **n-1** polynomial **R(y)** is determined by **$R(y_0)$, … $R(y_{n-1})$** for any **n** distinct **$y_0,\ldots,y_{n-1}$**

- To compute **PQ** (assume degree at most **n/2-1**)

  Evaluate **$P(y_0),\ldots, P(y_{n-1})$**

  Evaluate **$Q(y_0),\ldots,Q(y_{n-1})$**

  Multiply values **$P(y_i)Q(y_i)$** for **$i=0,\ldots,n-1$**

  Interpolate to recover **PQ**

Interpolation

- Given values of degree **n-1** polynomial **R** at **n** distinct points $y_0,\ldots,y_{n-1}$

  $R(y_0),\ldots,R(y_{n-1})$
- Compute coefficients $c_0,\ldots,c_{n-1}$ such that

  $R(x)=c_0+c_1x+c_2x^2+\ldots+c_{n-1}x^{n-1}$
- System of linear equations in $c_0,\ldots,c_{n-1}$

$$c_0+c_1y_0+c_2y_0^2+\ldots+c_{n-1}y_0^{n-1}=R(y_0)$$
$$c_0+c_1y_1+c_2y_1^2+\ldots+c_{n-1}y_1^{n-1}=R(y_1)$$
$$\ldots$$
$$c_0+c_1y_{n-1}+c_2y_{n-1}^2+\ldots+c_{n-1}y_{n-1}^{n-1}=R(y_{n-1})$$

known

unknown

Interpolation:
n equations in n unknowns

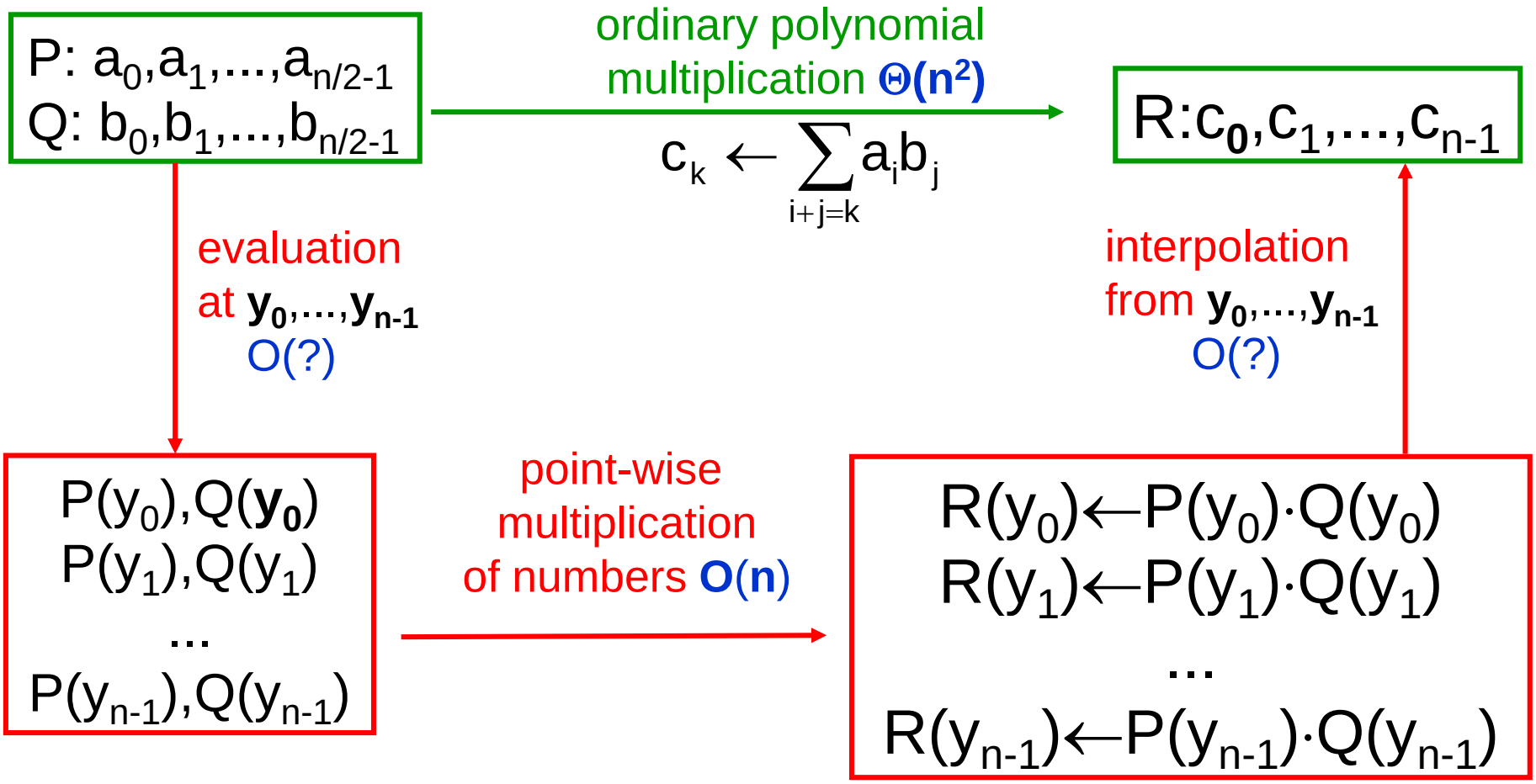
- Matrix form of the linear system

$$\begin{pmatrix} 1 & y_0 & y_0^2 & \cdots & y_0^{n-1} \\ 1 & y_1 & y_1^2 & \cdots & y_1^{n-1} \\ & \cdots & & & \\ & \cdots & & & \\ 1 & y_{n-1} & y_{n-1}^2 & \cdots & y_{n-1}^{n-1} \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ . \\ c_{n-1} \end{pmatrix} = \begin{pmatrix} R(y_0) \\ R(y_1) \\ . \\ . \\ R(y_{n-1}) \end{pmatrix}$$

- Fact: Determinant of the matrix is $\prod_{i<j} (y_i - y_j)$ which is not **0** since points are distinct

  System has a unique solution $c_0, \ldots, c_{n-1}$

Evaluation & Interpolation



Karatsuba's algorithm and evaluation and interpolation

- Strassen gave a way of doing **2x2** matrix multiplies with fewer multiplications
- Karatsuba's algorithm can be thought of as a way of multiplying degree **1** polynomials (which have **2** coefficients) using fewer multiplications

$PQ=(P_0+P_1z)(Q_0+Q_1z)$

$= P_0Q_0 + (P_1Q_0+P_0Q_1)z + P_1Q_1z^2$

Evaluate at **0,1,-1** (Could also use other points)

- $A = P(0)Q(0)= P_0Q_0$
- $C = P(1)Q(1)=(P_0+P_1)(Q_0+Q_1)$
- $D = P(-1)Q(-1)=(P_0 -P_1)(Q_0 -Q_1)$

Interpolating, Karatsuba's $Mid=(C-D)/2$ and $B=(C+D)/2-A$

Evaluation at Special Points

- Evaluation of polynomial at **1** point takes **O**(**n**) time
  - So **2n** points (naively) takes **O**($\mathbf{n^2}$)—no savings
  - But the algorithm works no matter what the points are…
- So…choose points that are related to each other so that evaluation problems can share subproblems

The key idea: Evaluate at related points

- $P(x) = a_0+a_1x+a_2x^2+a_3x^3+a_4x^4+\ldots+a_{n-1}x^{n-1}$

  $= a_0 +a_2x^2 +a_4x^4 +\ldots+ a_{n-2}x^{n-2}$

  $+ a_1x+a_3x^3 +a_5x^5 +\ldots+a_{n-1}x^{n-1}$

  $= P_{even}(x^2) + x\, P_{odd}(x^2)$

- $P(-x)=a_0 -a_1x+a_2x^2 -a_3x^3+a_4x^4-\ldots -a_{n-1}x^{n-1}$

  $= a_0 +a_2x^2 +a_4x^4 +\ldots+ a_{n-2}x^{n-2}$

  $- (a_1x+a_3x^3 +a_5x^5 +\ldots+a_{n-1}x^{n-1})$

  $= P_{even}(x^2) - x\, P_{odd}(x^2)$

where $P_{even}(z) = a_0 +a_2z +a_4z^2 +\ldots+ a_{n-2}z^{n/2-1}$

and $P_{odd}(z) = a_1+a_3z +a_5z^2 +\ldots+a_{n-1}z^{n/2-1}$

The key idea: Evaluate at related points

- So… if we have half the points as negatives of the other half

  i.e., $y_{n/2} = -y_0$, $y_{n/2+1} = -y_1, \ldots, y_{n-1} = -y_{n/2-1}$

  then we can reduce the size **n** problem of evaluating degree **n-1** polynomial **P** at **n** points to evaluating **2** degree **n/2 - 1** polynomials $\mathbf{P_{even}}$ and $\mathbf{P_{odd}}$ at **n/2** points $y_0^2, \ldots y_{n/2-1}^2$ and recombine answers with **O(1)** extra work per point

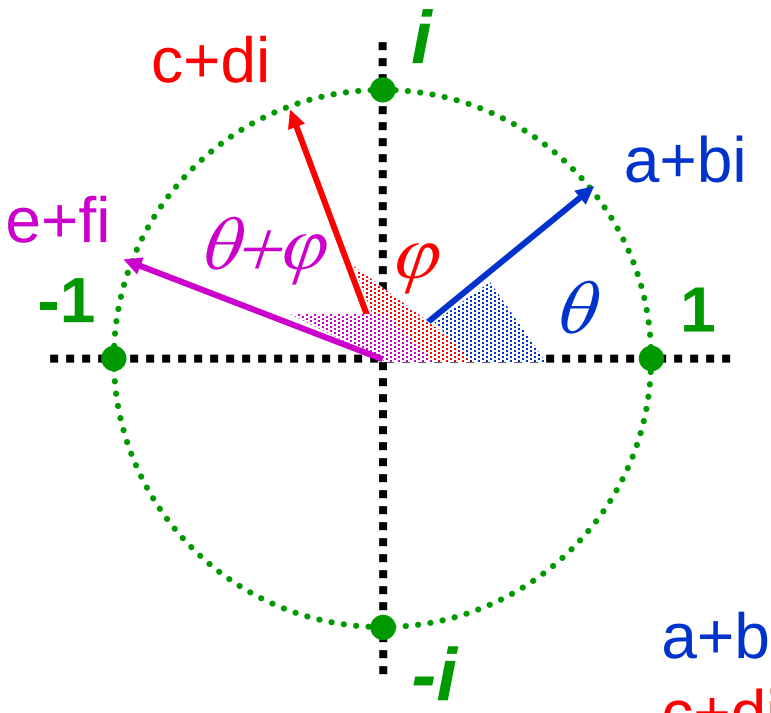
- But to use this idea recursively we need half of $y_0^2, \ldots y_{n/2-1}^2$ to be negatives of the other half

  If $y_{n/4}^2 = -y_0^2$**,** say, then $(y_{n/4}/y_0)^2 = -1$

  Motivates use of complex numbers as evaluation points

Complex Numbers

$i^2 = -1$



To multiply complex numbers:
1. add angles
2. multiply lengths
(all length 1 here)

$e+fi = (a+bi)(c+di)$

$a+bi = \cos\theta + i\sin\theta = e^{i\theta}$

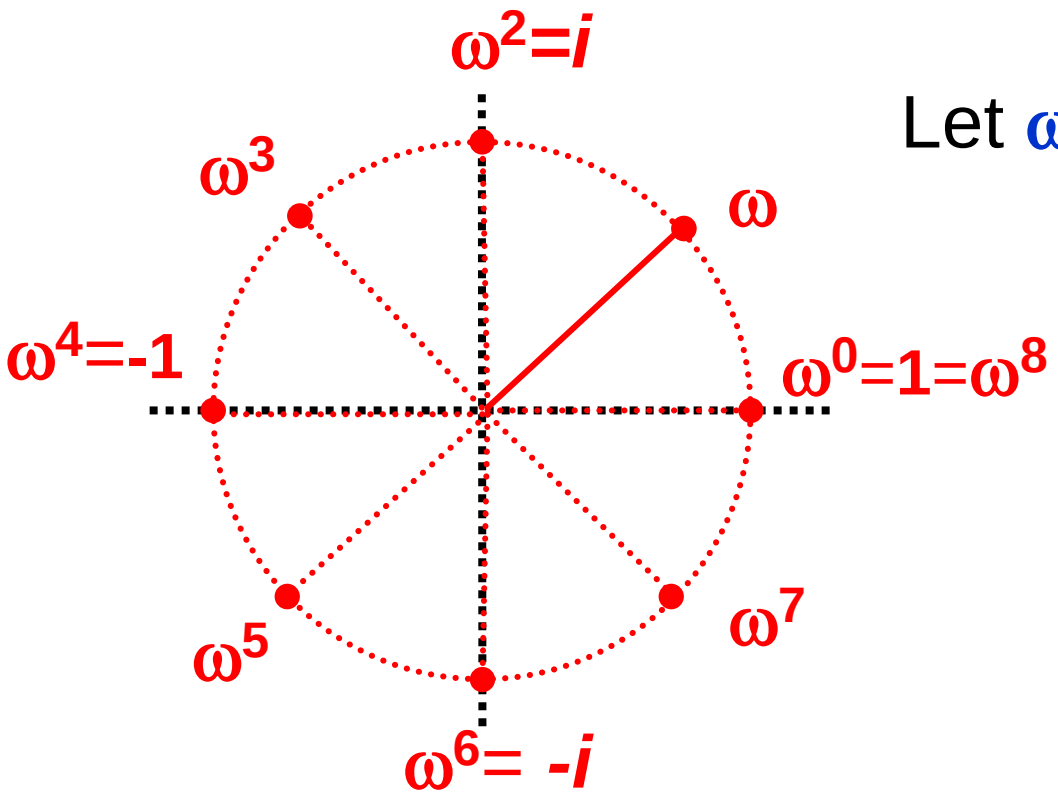
$c+di = \cos\varphi + i\sin\varphi = e^{i\varphi}$

$e+fi = \cos(\theta+\varphi) + i\sin(\theta+\varphi) = e^{i(\theta+\varphi)}$

$e^{2\pi i} = 1$

$e^{\pi i} = -1$

Primitive n^{th} root of 1 $\omega = \omega_n = e^{i 2\pi/n}$



Let $\omega = \omega_n = e^{i\,2\pi\,/n}$

$= \cos(2\pi/n) + i \sin(2\pi/n)$

$i^2 = -1$

$e^{2\pi\, i} = 1$

Facts about $\omega = e^{2\pi i / n}$ for even n

- $\omega = e^{2\pi i / n}$ for $i = \sqrt{-1}$
- $\omega^n = 1$
- $\omega^{n/2} = -1$
- $\omega^{n/2+k} = -\omega^k$ for all values of **k**
- $\omega^2 = e^{2\pi i / m}$ where $m = n/2$
- $\omega^k = \cos(2k\pi/n) + i \sin(2k\pi/n)$ so can compute with powers of ω
- ω^k is a root of $x^n - 1 = (x-1)(x^{n-1}+x^{n-2}+\ldots+1) = 0$ but for $k \neq 0$, $\omega^k \neq 1$ so $\omega^{k(n-1)}+\omega^{k(n-2)}+\ldots+1 = 0$

The key idea for n even

- $P(\omega) = a_0+a_1\omega+a_2\omega^2+a_3\omega^3+a_4\omega^4+\ldots+a_{n-1}\omega^{n-1}$

  $= a_0 +a_2\omega^2 +a_4\omega^4 +\ldots+ a_{n-2}\omega^{n-2}$

  $+ a_1\omega+a_3\omega^3 +a_5\omega^5 +\ldots+a_{n-1}\omega^{n-1}$

  $= P_{even}(\omega^2) + \omega\, P_{odd}(\omega^2)$

- $P(-\omega)=a_0 -a_1\omega+a_2\omega^2 -a_3\omega^3+a_4\omega^4-\ldots -a_{n-1}\omega^{n-1}$

  $= a_0 +a_2\omega^2 +a_4\omega^4 +\ldots+ a_{n-2}\omega^{n-2}$

  $- (a_1\omega+a_3\omega^3 +a_5\omega^5 +\ldots+a_{n-1}\omega^{n-1})$

  $= P_{even}(\omega^2) - \omega\, P_{odd}(\omega^2)$

where $P_{even}(x) = a_0 +a_2x +a_4x^2 +\ldots+ a_{n-2}x^{n/2-1}$

and $P_{odd}(x) = a_1+a_3x +a_5x^2 +\ldots+a_{n-1}x^{n/2-1}$

The recursive idea for n a power of 2

- Goal:

  Evaluate **P** at $\mathbf{1},\omega,\omega^2,\omega^3,\ldots,\omega^{n-1}$

- Now

  P_{even} and P_{odd} have degree $n/2-1$ where

  $P(\omega^k)=P_{even}(\omega^{2k})+\omega^k P_{odd}(\omega^{2k})$

  $P(-\omega^k)=P_{even}(\omega^{2k})-\omega^k P_{odd}(\omega^{2k})$

- Recursive Algorithm

  Evaluate P_{even} at $\mathbf{1},\omega^2,\omega^4,\ldots,\omega^{n-2}$

  Evaluate P_{odd} at $\mathbf{1},\omega^2,\omega^4,\ldots,\omega^{n-2}$

  Combine to compute **P** at $\mathbf{1},\omega,\omega^2,\ldots,\omega^{n/2-1}$

  Combine to compute **P** at $\mathbf{-1},-\omega,-\omega^2,\ldots,-\omega^{n/2-1}$
  (i.e. at $\omega^{n/2}$, $\omega^{n/2+1}$, $\omega^{n/2+2}$,…, ω^{n-1})

ω^2 is $e^{2\pi i/m}$ where **m=n/2** so problems are of same type but smaller size

Analysis and more

- Run-time

  $T(n)=2\cdot T(n/2)+cn$ so $T(n)=O(n \log n)$

- So much for evaluation ... what about interpolation?

  Given

  - $r_0=R(1)$, $r_1=R(\omega)$, $r_2=R(\omega^2)$,…, $r_{n-1}=R(\omega^{n-1})$

  Compute

  - $c_0, c_1,\ldots,c_{n-1}$ s.t. $R(x)=c_0+c_1x+\ldots+c_{n-1}x^{n-1}$

Interpolation ≈ Evaluation: strange but true

- Non-obvious fact:

  If we define a new polynomial
  $S(x) = r_0 + r_1x + r_2x^2 + \ldots + r_{n-1}x^{n-1}$ where $r_0, r_1, \ldots, r_{n-1}$ are the evaluations of **R** at $1, \omega, \ldots, \omega^{n-1}$

  Then $c_k = S(\omega^{-k})/n$ for $k=0,\ldots,n-1$

  Relies on the fact the interpolation (inverse) matrix has **ij** entry $\omega^{-(i+j)}/n$ instead of ω^{i+j}

- So...

  evaluate **S** at $1,\omega^{-1},\omega^{-2},\ldots,\omega^{-(n-1)}$ then divide each answer by **n** to get the $c_0,\ldots,c_{n-1}$

  ω^{-1} behaves just like ω did so the same $O(n \log n)$ evaluation algorithm applies !