Mathematics in OI (III)

Anson Ho

1 July 2019

Mathematics in OI (I)

- Modular arithmetic
  (including modular inverse and Euler's Totient Theorem)

Mathematics in OI (I)

- Modular arithmetic
  (including modular inverse and Euler's Totient Theorem)
- Fast exponentiation

Mathematics in OI (I)

- Modular arithmetic
  (including modular inverse and Euler's Totient Theorem)
- Fast exponentiation
- High precision arithmetic (HPA)

Mathematics in OI (I)

- Modular arithmetic
  (including modular inverse and Euler's Totient Theorem)
- Fast exponentiation
- High precision arithmetic (HPA)
- Greatest common divisor (GCD)

Mathematics in OI (I)

- Modular arithmetic
  (including modular inverse and Euler's Totient Theorem)
- Fast exponentiation
- High precision arithmetic (HPA)
- Greatest common divisor (GCD)
- Extended Euclidean Algorithm

Mathematics in OI (I)

- Modular arithmetic
  (including modular inverse and Euler's Totient Theorem)
- Fast exponentiation
- High precision arithmetic (HPA)
- Greatest common divisor (GCD)
- Extended Euclidean Algorithm
- Sieve of Eratosthenes

Mathematics in OI (II)

- Permutation and combination (P_r^n and C_r^n)

Mathematics in OI (II)

- Permutation and combination (P_r^n and C_r^n)
- Stars and Bars (H_r^n)

Mathematics in OI (II)

- Permutation and combination (P_r^n and C_r^n)
- Stars and Bars (H_r^n)
- Chinese Remainder Theorem (CRT)

Mathematics in OI (II)

- Permutation and combination (P_r^n and C_r^n)
- Stars and Bars (H_r^n)
- Chinese Remainder Theorem (CRT)
- Fibonacci Number and Catalan Number (and its variant)

Mathematics in OI (II)

- Permutation and combination (P_r^n and C_r^n)
- Stars and Bars (H_r^n)
- Chinese Remainder Theorem (CRT)
- Fibonacci Number and Catalan Number (and its variant)
- Inclusion-exclusion principle

Mathematics in OI (II)

- Permutation and combination (P_r^n and C_r^n)
- Stars and Bars (H_r^n)
- Chinese Remainder Theorem (CRT)
- Fibonacci Number and Catalan Number (and its variant)
- Inclusion-exclusion principle
- Elementary Probability

Mathematics in OI (II)

- Permutation and combination (P_r^n and C_r^n)
- Stars and Bars (H_r^n)
- Chinese Remainder Theorem (CRT)
- Fibonacci Number and Catalan Number (and its variant)
- Inclusion-exclusion principle
- Elementary Probability
- Expectation and counting

Mathematics in OI (III)

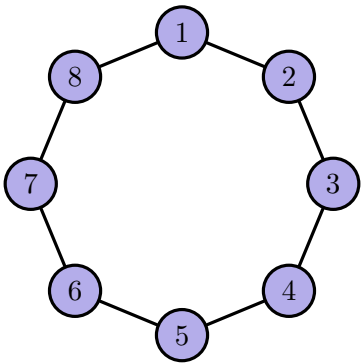
- Josephus Problem
- Gaussian Elimination
- Matrix multiplication
- Lagrange Interpolation
- Cayley's Formula and Prufer sequence

Josephus Problem

- Related tasks: HKOI 01030 01043
- N people numbered $1, 2, \ldots, N$ are standing in a circle
- Start counting from 1
- Skip $K - 1$ people and then kill then K^{th} person
- Repeat the above step until only one person left
- Find the id of the survivor

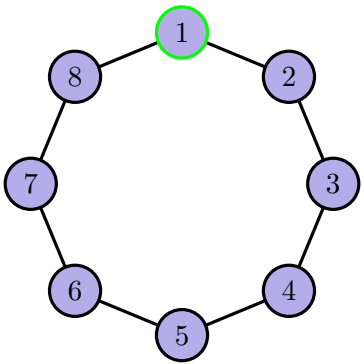
Josephus Problem

$N = 8, K = 3$



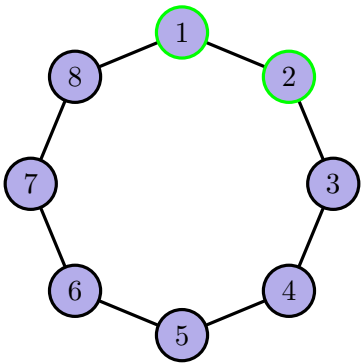
Josephus Problem

$N = 8, K = 3$



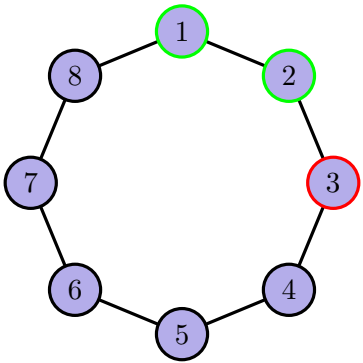
Josephus Problem

$N = 8, K = 3$



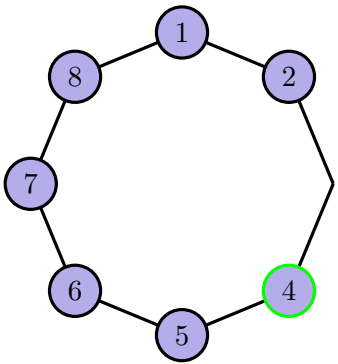
Josephus Problem

$N = 8, K = 3$



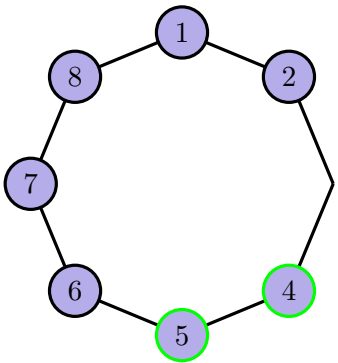
Josephus Problem

$N = 8, K = 3$



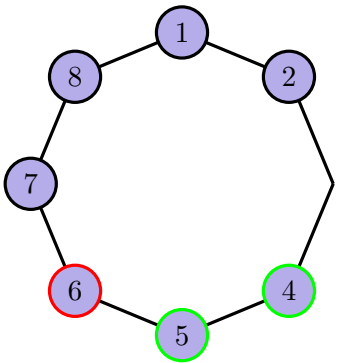
Josephus Problem

$N = 8, K = 3$



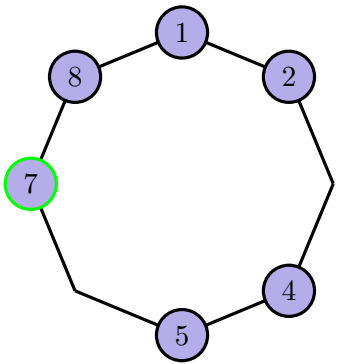
Josephus Problem

$N = 8, K = 3$



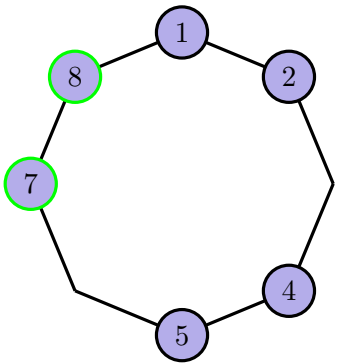
Josephus Problem

$N = 8, K = 3$



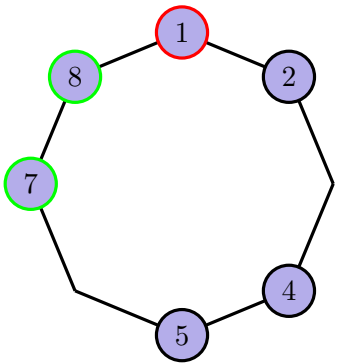
Josephus Problem

$N = 8, K = 3$



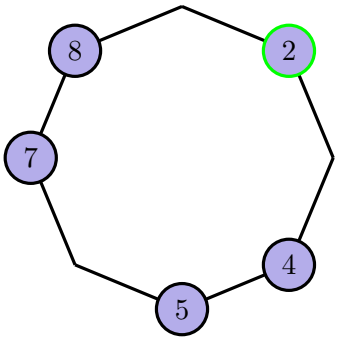
Josephus Problem

$N = 8, K = 3$



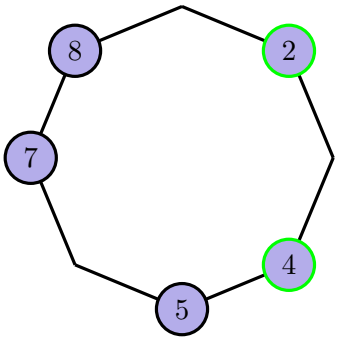
Josephus Problem

$N = 8, K = 3$



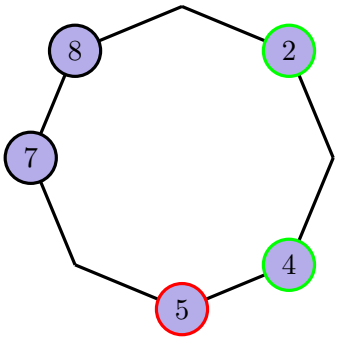
Josephus Problem

$N = 8, K = 3$



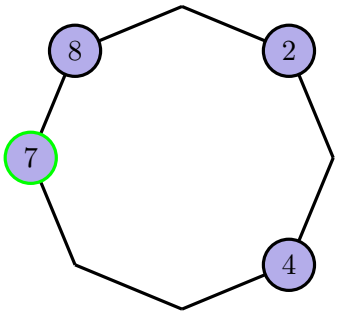
Josephus Problem

$N = 8, K = 3$



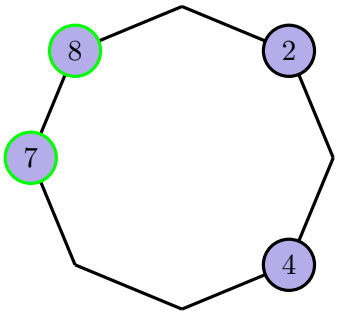
Josephus Problem

$N = 8, K = 3$



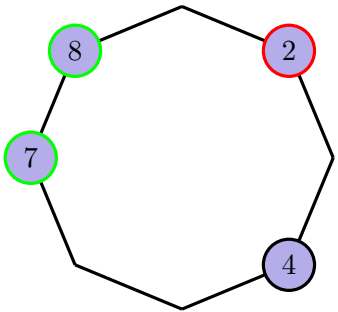
Josephus Problem

$N = 8, K = 3$



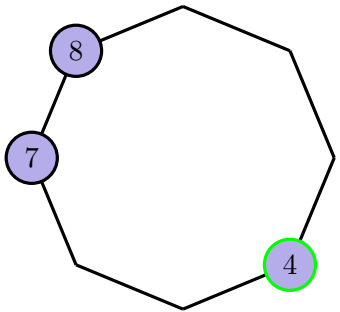
Josephus Problem

$N = 8, K = 3$



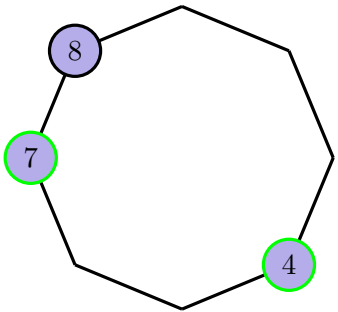
Josephus Problem

$N = 8, K = 3$



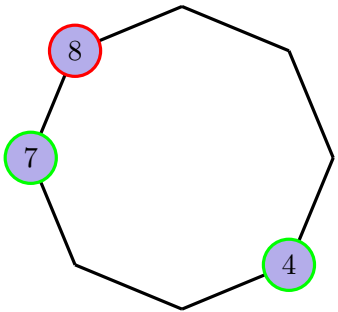
Josephus Problem

$N = 8, K = 3$



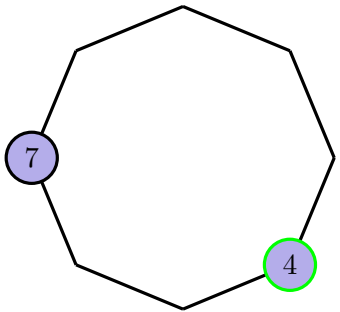
Josephus Problem

$N = 8, K = 3$



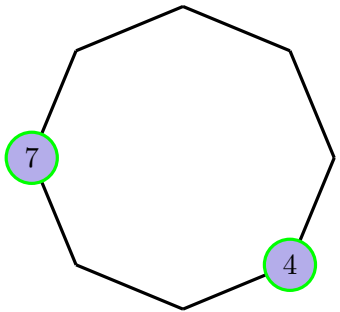
Josephus Problem

$N = 8, K = 3$



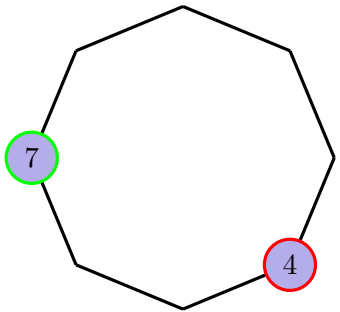
Josephus Problem

$N = 8, K = 3$



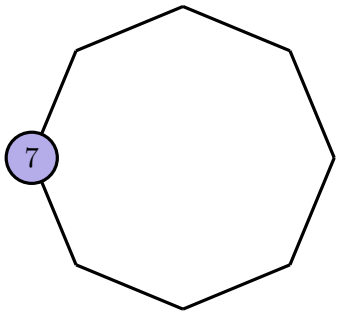
Josephus Problem

$N = 8, K = 3$



Josephus Problem

$N = 8, K = 3$



Josephus Problem

- Naive implementation: O(NK)

```
def josephus(n, k):
  alive = [i for i in range(1, n + 1)]
  while len(alive) > 1:
    for i in range(k - 1):
      alive.append(alive[0])
      del alive[0]
    del alive[0]
  return alive[0]
```

- Want to achieve a better time complexity

Josephus Problem - Pattern

N \ K	1	2	3	4	5	6	7	8	9
1	1	1	1	1	1	1	1	1	1
2	2	1	2	1	2	1	2	1	2
3	3	3	2	2	1	1	3	3	2
4	4	1	1	2	2	3	2	3	3
5	5	3	4	1	2	4	4	1	2
6	6	5	1	5	1	4	5	3	5
7	7	7	4	2	6	3	5	4	7
8	8	1	7	6	3	1	4	4	8
9	9	3	1	1	8	7	2	3	8

Josephus Problem - Pattern

N \ K	1	2	3	4	5	6	7	8	9
1	1	1	1	1	1	1	1	1	1
2	2	1	2	1	2	1	2	1	2
3	3	3							
4	4	1							
5	5	3							
6	6	5							
7	7	7							
8	8	1							
9	9	3							

Josephus Problem - Pattern

N \ K	1	2	3	4	5	6	7	8	9
1	1	1	1						
2	2	1	2						
3	3	3	2						
4	4	1	1						
5	5	3	4						
6	6	5	1						
7	7	7	4						
8	8	1	7						
9	9	3	1						

Josephus Problem - Pattern

N \ K	1	2	3	4	5	6	7	8	9
1	1	1	1						
2	2	1	2						
3	3	3	2						
4	4	1	1						
5	5	3	4						
6	6	5	1						
7	7	7	4						
8	8	1	7						
9	9	3	1						

Josephus Problem - Pattern

N \ K	1	2	3	4	5	6	7	8	9
1	1	1	1						
2	2	1	4						
3	3	3	2						
4	4	1	1						
5	5	3	4						
6	6	5	1						
7	7	7	4						
8	8	1	7						
9	9	3	1						

Josephus Problem - Pattern

N \ K	1	2	3	4	5	6	7	8	9
1	1	1	1						
2	2	1	2						
3	3	3	2						
4	4	1	1						
5	5	3	4						
6	6	5	1						
7	7	7	4						
8	8	1	7						
9	9	3	1						

Josephus Problem - Pattern

N \ K	1	2	3	4	5	6	7	8	9
1	1	1	1						
2	2	1	2						
3	3	3	5						
4	4	1	1						
5	5	3	4						
6	6	5	1						
7	7	7	4						
8	8	1	7						
9	9	3	1						

Josephus Problem - Pattern

N \ K	1	2	3	4	5	6	7	8	9
1	1	1	1						
2	2	1	2						
3	3	3	2						
4	4	1	1						
5	5	3	4						
6	6	5	1						
7	7	7	4						
8	8	1	7						
9	9	3	1						

Josephus Problem - Pattern

N \ K	1	2	3	4	5	6	7	8	9
1	1	1	1						
2	2	1	2						
3	3	3	2						
4	4	1	5						
5	5	3	4						
6	6	5	1						
7	7	7	4						
8	8	1	7						
9	9	3	1						

Josephus Problem - Pattern

N \ K	1	2	3	4	5	6	7	8	9
1	1	1	1	1	1	1	1	1	1
2	2	1	2	1	2	1	2	1	2
3	3	3	2	2	1	1	3	3	2
4	4	1	1	2	2	3	2	3	3
5	5	3	4	1	2	4	4	1	2
6	6	5	1	5	1	4	5	3	5
7	7	7	4	2	6	3	5	4	7
8	8	1	7	6	3	1	4	4	8
9	9	3	1	1	8	7	2	3	8

Josephus Problem - Formula

$$josephus(N, K) = \begin{cases} 1 & N = 1 \\ ? & \text{otherwise} \end{cases}$$

Josephus Problem - Formula

$$josephus(N, K) = \begin{cases} 1 & N = 1 \\ josephus(N-1, K) + K & \text{otherwise} \end{cases}$$

Josephus Problem - Formula

$$josephus(N, K) = \begin{cases} 1 & N = 1 \\ [josephus(N-1, K) + K - 1] \bmod N + 1 & \text{otherwise} \end{cases}$$

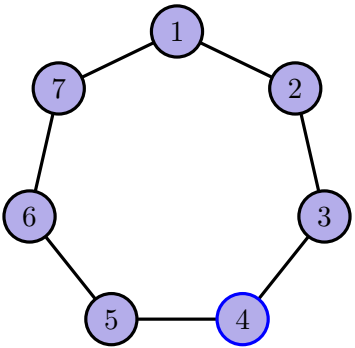
Josephus Problem - Formula

(Assume 0-based index is used)

$$josephus(N, K) = \begin{cases} 0 & N = 1 \\ [josephus(N-1, K) + K] \bmod N & \text{otherwise} \end{cases}$$

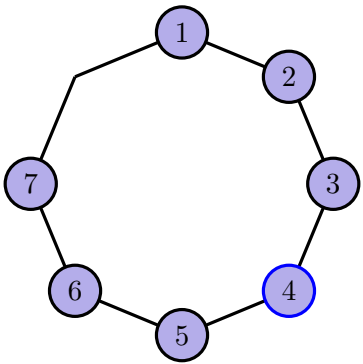
Josephus Problem

$N = 7, K = 3$



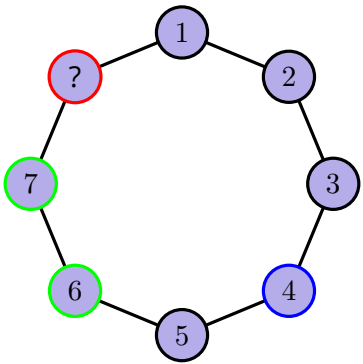
Josephus Problem

$N = 7, K = 3$



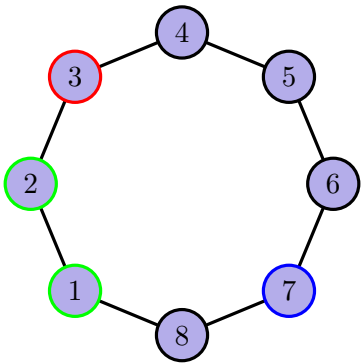
Josephus Problem

$N = 8, K = 3$



Josephus Problem

$N = 8, K = 3$



Josephus Problem

- Revision on modular arithmetic

Josephus Problem

- Revision on modular arithmetic
- Generating small cases sometimes gives you some insights

Josephus Problem

- Revision on modular arithmetic
- Generating small cases sometimes gives you some insights
- Try to tackle problem in other direction

Gaussian Elimination

- A systematic way to solve simultaneous linear equations

$$\begin{cases} c_{1,1}x_1 + c_{1,2}x_2 + \cdots + c_{1,m}x_m = d_1 \\ c_{2,1}x_1 + c_{2,2}x_2 + \cdots + c_{2,m}x_m = d_2 \\ \vdots \\ c_{n,1}x_1 + c_{n,2}x_2 + \cdots + c_{n,m}x_m = d_n \end{cases}$$

where $x_1, x_2, \ldots, x_m$ are variables and the others are constants

Gaussian Elimination

- A systematic way to solve simultaneous linear equations

$$\begin{cases} c_{1,1}x_1 + c_{1,2}x_2 + \cdots + c_{1,m}x_m = d_1 \\ c_{2,1}x_1 + c_{2,2}x_2 + \cdots + c_{2,m}x_m = d_2 \\ \vdots \\ c_{n,1}x_1 + c_{n,2}x_2 + \cdots + c_{n,m}x_m = d_n \end{cases}$$

  where $x_1, x_2, \ldots, x_m$ are variables and the others are constants

- Actually, it is just a generalization of what you learnt in Mathematics lessons

Gaussian Elimination - Operations

- Multiply a equation by a constant

$$c_{i,1}x_1 + c_{i,2}x_2 + \cdots + c_{i,m}x_m = d_i$$
$$\Downarrow$$
$$kc_{i,1}x_1 + kc_{i,2}x_2 + \cdots + kc_{i,m}x_m = kd_i$$

Gaussian Elimination - Operations

- Swap two equations

$$
\begin{aligned}
c_{i,1}x_1 + c_{i,2}x_2 + \cdots + c_{i,m}x_m &= d_i \\
c_{j,1}x_1 + c_{j,2}x_2 + \cdots + c_{j,m}x_m &= d_j \\
&\Downarrow \\
c_{j,1}x_1 + c_{j,2}x_2 + \cdots + c_{j,m}x_m &= d_j \\
c_{i,1}x_1 + c_{i,2}x_2 + \cdots + c_{i,m}x_m &= d_i
\end{aligned}
$$

Gaussian Elimination - Operations

- Add a multiple of a equation to another

$$
\begin{aligned}
c_{i,1}x_1 + c_{i,2}x_2 + \cdots + c_{i,m}x_m &= d_i \\
c_{j,1}x_1 + c_{j,2}x_2 + \cdots + c_{j,m}x_m &= d_j \\
\Downarrow & \\
(c_{i,1} + kc_{j,1})x_1 + (c_{i,2} + kc_{j,2})x_2 + \cdots + (c_{i,m} + kc_{j,m})x_m &= d_i + kd_j \\
c_{j,1}x_1 + c_{j,2}x_2 + \cdots + c_{j,m}x_m &= d_j
\end{aligned}
$$

Gaussian Elimination - Example

$$
\begin{aligned}
y + z &= 1 \\
x + 2y + z &= 3 \\
x + 3y &= 4
\end{aligned}
$$

Gaussian Elimination - Example

$$
\begin{aligned}
x + 2y + z &= 3 \\
y + z &= 1 \\
x + 3y &= 4
\end{aligned}
$$

- Swap two equations

Gaussian Elimination - Example

$$
\begin{aligned}
x + 2y + z &= 3 \\
y + z &= 1 \\
y - z &= 1
\end{aligned}
$$

- Add a multiple of a equation to another

Gaussian Elimination - Example

$$
\begin{aligned}
x + 2y + z &= 3 \\
y + z &= 1 \\
-2z &= 0
\end{aligned}
$$

- Add a multiple of a equation to another

Gaussian Elimination - Example

$$
\begin{aligned}
x + 2y + z &= 3 \\
y + z &= 1 \\
z &= 0
\end{aligned}
$$

- Multiply a equation by a constant

Gaussian Elimination - Example

$$
\begin{aligned}
x + 2y + z &= 3 \\
y + z &= 1 \\
z &= 0
\end{aligned}
$$

At this point, we know that it has unique solution

Gaussian Elimination - Example

$$
\begin{aligned}
x + 2y &= 3 \\
y + z &= 1 \\
z &= 0
\end{aligned}
$$

- Add a multiple of a equation to another

Gaussian Elimination - Example

$$
\begin{aligned}
x + 2y &= 3 \\
y &= 1 \\
z &= 0
\end{aligned}
$$

- Add a multiple of a equation to another

Gaussian Elimination - Example

$$x = 1$$
$$y = 1$$
$$z = 0$$

- Add a multiple of a equation to another

Gaussian Elimination - Procedure

1. Handle columns one by one
2. For each column, find a not fixed row
   such that the corresponding entry is non-zero
   (skip that column if none of the rows satisfies)
3. Fix that row
4. Multiply that row by a constant
   such that the corresponding entry becomes 1
5. Add a multiple of that row to each of the other rows
   (including the fixed ones)
   such that the corresponding entries in other rows become 0

Gaussian Elimination - Special cases

- No solution
  - The last entry of a row in the final status is the only non-zero entry in that row (e.g. $0 = 3$)
- More than one solution (some variables are not fixed)
  - Some columns are skipped (except the above case)

Gaussian Elimination - Implementation

- Time complexity: $O(N^3)$ for N equations and N unknowns
- Use a 2D array to store everything
- Common practice:
  - Handle columns from left to right
  - Swap the newly fixed row to the top among all not fixed rows

Gaussian Elimination - Implementation

- Fix the row with largest absolute value of the corresponding entry to reduce numerical error

Gaussian Elimination - Implementation

- Fix the row with largest absolute value of the corresponding entry to reduce numerical error
- To get exact value
  - Do all operations under a modular base (or)
  - Use fractions everywhere (or)
  - In the elimination stage (the last step of each coloumn), multiply the newly fixed row and the target row such that the two corresponding entries become their L.C.M. (then the corresponding entry of the newly fixed row is no longer necessary to be 1)

Gaussian Elimination - Implementation

- Fix the row with largest absolute value of the corresponding entry to reduce numerical error
- To get exact value
  - Do all operations under a modular base (or)
  - Use fractions everywhere (or)
  - In the elimination stage (the last step of each coloumn), multiply the newly fixed row and the target row such that the two corresponding entries become their L.C.M. (then the corresponding entry of the newly fixed row is no longer necessary to be 1)
- `std::bitset` can be used to optimize if the modular base is 2

Gaussian Elimination - Related tasks

- Codeforces `845G` (no. of AC $>>$ `E` and `F`)
  - Find xor shortest path given starting and ending vertex

Gaussian Elimination - Related tasks

- Codeforces `845G` (no. of AC $>>$ `E` and `F`)
  - Find xor shortest path given starting and ending vertex
  - The path length is defined by the xor sum of the weights of the edges

Gaussian Elimination - Related tasks

- Codeforces 845G (no. of AC >> E and F)
  - Find xor shortest path given starting and ending vertex
  - The path length is defined by the xor sum of the weights of the edges
  - Identify cycles by back edges in DFS

Gaussian Elimination - Related tasks

- Codeforces `845G` (no. of AC >> `E` and `F`)
  - Find xor shortest path given starting and ending vertex
  - The path length is defined by the xor sum of the weights of the edges
  - Identify cycles by back edges in DFS
  - Represent each cycle by a binary vector (its path sum in binary notation)

Gaussian Elimination - Related tasks

- Codeforces `845G` (no. of AC >> `E` and `F`)
  - Find xor shortest path given starting and ending vertex
  - The path length is defined by the xor sum of the weights of the edges
  - Identify cycles by back edges in DFS
  - Represent each cycle by a binary vector (its path sum in binary notation)
  - Find the basis of them by Gaussian Elimination

Gaussian Elimination - Related tasks

- Codeforces `845G` (no. of AC >> `E` and `F`)
  - Find xor shortest path given starting and ending vertex
  - The path length is defined by the xor sum of the weights of the edges
  - Identify cycles by back edges in DFS
  - Represent each cycle by a binary vector (its path sum in binary notation)
  - Find the basis of them by Gaussian Elimination
- Codeforces `1101G`
  - Also related to xor basis

Gaussian Elimination - Other application

- Determinant of a matrix
- Rank of a matrix
- Inverse of a matrix

Matrix multiplication - Introduction to matrix

Vector $\leftrightarrow$ 1D array

Matrix $\leftrightarrow$ 2D array

(Tensor $\leftrightarrow$ multidimensional array)

Matrix multiplication - Introduction to matrix

Vector $\leftrightarrow$ 1D array

Matrix $\leftrightarrow$ 2D array

(Tensor $\leftrightarrow$ multidimensional array)

- Dimension: `int A[N][M];` $\leftrightarrow$ $N \times M$ matrix
  - N is number of rows
  - M is number of columns
- Comparing to a pure 2D array, matrix has some algebraic properties
- Support addition, subtraction and multiplication (under some restriction)

Matrix multiplication - Introduction to matrix

- Addition and subtraction
  - Requirement: two matrices must have the same dimension
  - Entrywise operation
- Multiplication
  - Requirement: the second dimension of the first matrix must equal to the first dimension of the second matrix
  - $A \times B$ is valid iff their dimensions are in the form $n \times m$ and $m \times k$ respectively
  - Associativity: $(A \times B) \times C = A \times (B \times C)$
    - HKOI 01054 is based on this property

Matrix multiplication

$$\begin{pmatrix} a & b \\ c & d \\ e & f \end{pmatrix} \times \begin{pmatrix} g & h & i \\ j & k & l \end{pmatrix} = \begin{pmatrix} ag+bj & ah+bk & ai+bl \\ cg+dj & ch+dk & ci+dl \\ eg+fj & eh+fk & ei+fl \end{pmatrix}$$

Matrix multiplication

$$\begin{pmatrix} a & b \\ c & d \\ e & f \end{pmatrix} \times \begin{pmatrix} g & h & i \\ j & k & l \end{pmatrix} = \begin{pmatrix} ag+bj & ah+bk & ai+bl \\ cg+dj & ch+dk & ci+dl \\ eg+fj & eh+fk & ei+fl \end{pmatrix}$$

Matrix multiplication

$$\begin{pmatrix} a & b \\ c & d \\ e & f \end{pmatrix} \times \begin{pmatrix} g & h & i \\ j & k & l \end{pmatrix} = \begin{pmatrix} ag+bj & ah+bk & ai+bl \\ cg+dj & ch+dk & ci+dl \\ eg+fj & eh+fk & ei+fl \end{pmatrix}$$

Matrix multiplication

$$\begin{pmatrix} a & b \\ c & d \\ e & f \end{pmatrix} \times \begin{pmatrix} g & h & i \\ j & k & l \end{pmatrix} = \begin{pmatrix} ag+bj & ah+bk & ai+bl \\ cg+dj & ch+dk & ci+dl \\ eg+fj & eh+fk & ei+fl \end{pmatrix}$$

Matrix multiplication

$$\begin{pmatrix} a & b \\ c & d \\ e & f \end{pmatrix} \times \begin{pmatrix} g & h & i \\ j & k & l \end{pmatrix} = \begin{pmatrix} ag+bj & ah+bk & ai+bl \\ cg+dj & ch+dk & ci+dl \\ eg+fj & eh+fk & ei+fl \end{pmatrix}$$

Matrix multiplication

$$\begin{pmatrix} a & b \\ c & d \\ e & f \end{pmatrix} \times \begin{pmatrix} g & h & i \\ j & k & l \end{pmatrix} = \begin{pmatrix} ag+bj & ah+bk & ai+bl \\ cg+dj & ch+dk & ci+dl \\ eg+fj & eh+fk & ei+fl \end{pmatrix}$$

Matrix multiplication

$$\begin{pmatrix} a & b \\ c & d \\ e & f \end{pmatrix} \times \begin{pmatrix} g & h & i \\ j & k & l \end{pmatrix} = \begin{pmatrix} ag+bj & ah+bk & ai+bl \\ cg+dj & ch+dk & ci+dl \\ eg+fj & eh+fk & ei+fl \end{pmatrix}$$

Matrix multiplication

$$\begin{pmatrix} a & b \\ c & d \\ e & f \end{pmatrix} \times \begin{pmatrix} g & h & i \\ j & k & l \end{pmatrix} = \begin{pmatrix} ag+bj & ah+bk & ai+bl \\ cg+dj & ch+dk & ci+dl \\ eg+fj & eh+fk & ei+fl \end{pmatrix}$$

Matrix multiplication

$$\begin{pmatrix} a & b \\ c & d \\ e & f \end{pmatrix} \times \begin{pmatrix} g & h & i \\ j & k & l \end{pmatrix} = \begin{pmatrix} ag+bj & ah+bk & ai+bl \\ cg+dj & ch+dk & ci+dl \\ eg+fj & eh+fk & ei+fl \end{pmatrix}$$

Matrix multiplication - Formula

- Let $A \in M_{n \times m}, B \in M_{m \times k}, C \in M_{n \times k}$ with $A \times B = C$
- Denote the entry of a matirx X in the i^{th} row, j^{th} column by $X_{i,j}$
- Then we have $C_{i,j} = \sum_{l=1}^{m} A_{i,l} B_{l,j}$

Matrix multiplication - Implementation

```
int a[N][M], b[M][K], c[N][K];
// assume a[][] and b[][] are ready here
for (int i = 0; i < N; i++)
  for (int j = 0; j < K; j++) {
    c[i][j] = 0;
    for (int k = 0; k < M; k++)
      c[i][j] += a[i][k] * b[k][j];
  }
```

- Time complexity: O(NMK)
- Swap the two inner for loop may optimize the performance a bit
  https://stackoverflow.com/questions/20467117/for-matrix-operation-why-is-ikj-faster-than-ijk

Matrix multiplication - System of linear equations

$$\begin{cases} c_{1,1}x_1 + c_{1,2}x_2 + \cdots + c_{1,m}x_m = d_1 \\ c_{2,1}x_1 + c_{2,2}x_2 + \cdots + c_{2,m}x_m = d_2 \\ \vdots \\ c_{n,1}x_1 + c_{n,2}x_2 + \cdots + c_{n,m}x_m = d_n \end{cases}$$

$$\Leftrightarrow \begin{pmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,m} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n,1} & c_{n,2} & \cdots & c_{n,m} \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{pmatrix}$$

Matrix multiplication - System of linear equations

$$\begin{cases} c_{1,1}x_1 + c_{1,2}x_2 + \cdots + c_{1,m}x_m = d_1 \\ c_{2,1}x_1 + c_{2,2}x_2 + \cdots + c_{2,m}x_m = d_2 \\ \vdots \\ c_{n,1}x_1 + c_{n,2}x_2 + \cdots + c_{n,m}x_m = d_n \end{cases}$$

$$\Leftrightarrow \begin{pmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,m} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n,1} & c_{n,2} & \cdots & c_{n,m} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{pmatrix}$$

Matrix multiplication - System of linear equations

$$
\begin{cases}
c_{1,1}x_1 + c_{1,2}x_2 + \cdots + c_{1,m}x_m = d_1 \\
c_{2,1}x_1 + c_{2,2}x_2 + \cdots + c_{2,m}x_m = d_2 \\
\vdots \\
c_{n,1}x_1 + c_{n,2}x_2 + \cdots + c_{n,m}x_m = d_n
\end{cases}
$$

$$
\Leftrightarrow
\begin{pmatrix}
c_{1,1} & c_{1,2} & \cdots & c_{1,m} \\
c_{2,1} & c_{2,2} & \cdots & c_{2,m} \\
\vdots & \vdots & \ddots & \vdots \\
c_{n,1} & c_{n,2} & \cdots & c_{n,m}
\end{pmatrix}
\begin{pmatrix}
x_1 \\ x_2 \\ \vdots \\ x_m
\end{pmatrix}
=
\begin{pmatrix}
d_1 \\ d_2 \\ \vdots \\ d_n
\end{pmatrix}
$$

Matrix multiplication - System of linear equations

$$
\begin{cases}
c_{1,1}x_1 + c_{1,2}x_2 + \cdots + c_{1,m}x_m = d_1 \\
c_{2,1}x_1 + c_{2,2}x_2 + \cdots + c_{2,m}x_m = d_2 \\
\vdots \\
c_{n,1}x_1 + c_{n,2}x_2 + \cdots + c_{n,m}x_m = d_n
\end{cases}
$$

$$
\Leftrightarrow
\begin{pmatrix}
c_{1,1} & c_{1,2} & \cdots & c_{1,m} \\
c_{2,1} & c_{2,2} & \cdots & c_{2,m} \\
\vdots & \vdots & \ddots & \vdots \\
c_{n,1} & c_{n,2} & \cdots & c_{n,m}
\end{pmatrix}
\begin{pmatrix}
x_1 \\ x_2 \\ \vdots \\ x_m
\end{pmatrix}
=
\begin{pmatrix}
d_1 \\ d_2 \\ \vdots \\ d_n
\end{pmatrix}
$$

Matrix multiplication - System of linear equations

$$\begin{cases} c_{1,1}x_1 + c_{1,2}x_2 + \cdots + c_{1,m}x_m = d_1 \\ c_{2,1}x_1 + c_{2,2}x_2 + \cdots + c_{2,m}x_m = d_2 \\ \vdots \\ c_{n,1}x_1 + c_{n,2}x_2 + \cdots + c_{n,m}x_m = d_n \end{cases}$$

$$\Leftrightarrow \begin{pmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,m} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n,1} & c_{n,2} & \cdots & c_{n,m} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{pmatrix}$$

Matrix multiplication - Transition matrix

- In some dynamic programming problems, the transition formula is a linear function of the previous state
- For example, $\begin{cases} \texttt{dp}[n+1][1] = \texttt{dp}[n][1] + 2\texttt{dp}[n][2] \\ \texttt{dp}[n+1][2] = 3\texttt{dp}[n][1] \end{cases}$
- Then it can be written as $\begin{pmatrix} 1 & 2 \\ 3 & 0 \end{pmatrix} \begin{pmatrix} \texttt{dp}[n][1] \\ \texttt{dp}[n][2] \end{pmatrix} = \begin{pmatrix} \texttt{dp}[n+1][1] \\ \texttt{dp}[n+1][2] \end{pmatrix}$

Matrix multiplication - Transition matrix

- In some dynamic programming problems, the transition formula is a linear function of the previous state
- For example, $\begin{cases} \texttt{dp}[n+1][1] = \texttt{dp}[n][1] + 2\texttt{dp}[n][2] \\ \texttt{dp}[n+1][2] = 3\texttt{dp}[n][1] \end{cases}$
- Then it can be written as $\begin{pmatrix} 1 & 2 \\ 3 & 0 \end{pmatrix} \begin{pmatrix} \texttt{dp}[n][1] \\ \texttt{dp}[n][2] \end{pmatrix} = \begin{pmatrix} \texttt{dp}[n+1][1] \\ \texttt{dp}[n+1][2] \end{pmatrix}$

Matrix multiplication - Transition matrix

- In some dynamic programming problems, the transition formula is a linear function of the previous state
- For example, $\begin{cases} \texttt{dp}[n+1][1] = \texttt{dp}[n][1] + 2\texttt{dp}[n][2] \\ \texttt{dp}[n+1][2] = 3\texttt{dp}[n][1] \end{cases}$
- Then it can be written as $\begin{pmatrix} 1 & 2 \\ 3 & 0 \end{pmatrix} \begin{pmatrix} \texttt{dp}[n][1] \\ \texttt{dp}[n][2] \end{pmatrix} = \begin{pmatrix} \texttt{dp}[n+1][1] \\ \texttt{dp}[n+1][2] \end{pmatrix}$

Matrix multiplication - Fibonacci Number

- $F_0 = 0, F_1 = 1, F_{n+2} = F_{n+1} + F_n$
- Then it can be written as $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix} = \begin{pmatrix} F_{n+2} \\ F_{n+1} \end{pmatrix}$

Matrix multiplication - Fibonacci Number

- $F_0 = 0, F_1 = 1, F_{n+2} = F_{n+1} + F_n$
- Then it can be written as $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix} = \begin{pmatrix} F_{n+2} \\ F_{n+1} \end{pmatrix}$

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n \begin{pmatrix} F_1 \\ F_0 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n-1} \begin{pmatrix} F_2 \\ F_1 \end{pmatrix} = \cdots = \begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix}$$

- Then fast exponentiation can be used to compute F_n
  (and the n^{th} term of any linear recurrence relation, in gernal)
  - Replace integer multiplication by matrix multiplication

Matrix multiplication - Sum of powers

Target: compute $\sum_{k=1}^{n} k^p = 1^p + 2^p + \cdots + n^p$

Matrix multiplication - Sum of powers

Target: compute $\sum_{k=1}^{n} k^p = 1^p + 2^p + \cdots + n^p$

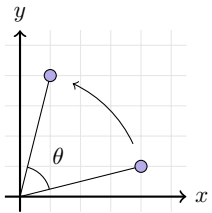
$$(n+1)^2 = n^2 + 2n + 1$$
$$(n+1)^3 = n^3 + 3n^2 + 3n + 1$$

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 3 & 3 & 1 \\ 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \sum_{k=1}^{n} k^3 \\ (n+1)^3 \\ (n+1)^2 \\ n+1 \\ 1 \end{pmatrix} = \begin{pmatrix} \sum_{k=1}^{n-1} k^3 \\ n^3 \\ n^2 \\ n \\ 1 \end{pmatrix}$$

Matrix multiplication - Affine transformation

- Rotation
  - Rotate counterclockwise through θ about the origin

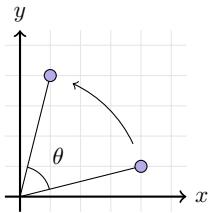
$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$



Matrix multiplication - Affine transformation

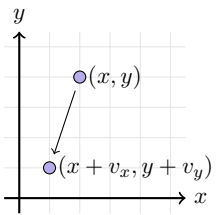
- Rotation
  - Rotate counterclockwise through θ about the origin

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$



- Translation
  - Translate by (v_x, v_y)

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & v_x \\ 0 & 1 & v_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$



Matrix multiplication - Related tasks

- HKOI `M1905`
- Codeforces `1151F`
- Codeforces `1182E`
- Codeforces `618E`

Lagrange Interpolation

- Target:
  - given $(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)$
  - find a polynomial $f(x)$ of degree $\leq k-1$ satisfying $y_i = f(x_i)$ for $i = 1, 2, \ldots, k$

Lagrange Interpolation

- Target:
  - given $(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)$
  - find a polynomial $f(x)$ of degree $\leq k-1$ satisfying $y_i = f(x_i)$ for $i = 1, 2, \ldots, k$

- Alternative solution: Gaussian elimination

$$\begin{pmatrix} x_1^{k-1} & x_1^{k-2} & \ldots & x_1 & 1 \\ x_2^{k-1} & x_2^{k-2} & \ldots & x_2 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_k^{k-1} & x_k^{k-2} & \ldots & x_k & 1 \end{pmatrix} \begin{pmatrix} a_{k-1} \\ a_{k-2} \\ \vdots \\ a_1 \\ a_0 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{pmatrix}$$

Then $f(x) = a_{k-1}x^{k-1} + a_{k-2}x^{k-2} + \cdots + a_1 x + a_0$

Lagrange Interpolation - Uniqueness of solution

- Target:
  - given $(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)$
  - find a polynomial $f(x)$ of degree $\leq k-1$ satisfying $y_i = f(x_i)$ for $i = 1, 2, \ldots, k$

- Suppose there is two solutions $f_1(x), f_2(x)$
  Let $p(x) = f_1(x) - f_2(x)$
  Note $p(x_i) = f_1(x_i) - f_2(x_i) = y_i - y_i = 0$ for $i = 1, 2, \ldots, k$
  Then $p(x) = q(x)(x - x_1)(x - x_2)\ldots(x - x_k)$
  $\deg q = \deg p - k \leq (k-1) - k < 0$
  $\therefore p(x) = q(x) = 0$

Lagrange Interpolation - Basis

- Consider $(1,0),(2,1),(3,0),(4,0),\ldots,(k,0)$

$$f(x) = p(x)(x-1)(x-3)(x-4)\ldots(x-k)$$
$$1 = f(2) = p(x)(2-1)(2-3)(2-4)\ldots(2-k)$$

Lagrange Interpolation - Basis

- Consider $(1,0),(2,1),(3,0),(4,0),\ldots,(k,0)$

$$f(x) = p(x)(x-1)(x-3)(x-4)\ldots(x-k)$$
$$1 = f(2) = p(x)(2-1)(2-3)(2-4)\ldots(2-k)$$

- Take $p(x) = \frac{1}{(2-1)(2-3)(2-4)\ldots(2-k)}$
  Then $f(x) = \frac{(x-1)(x-3)(x-4)\ldots(x-k)}{(2-1)(2-3)(2-4)\ldots(2-k)}$

Lagrange Interpolation - Basis

- Consider $(1,0),(2,1),(3,0),(4,0),\ldots,(k,0)$
  Then $f(x)=\frac{(x-1)(x-3)(x-4)\ldots(x-k)}{(2-1)(2-3)(2-4)\ldots(2-k)}$

Lagrange Interpolation - Basis

- Consider $(1,0),(2,1),(3,0),(4,0),\ldots,(k,0)$
  Then $f(x)=\frac{(x-1)(x-3)(x-4)...(x-k)}{(2-1)(2-3)(2-4)...(2-k)}$
- Consider $(x_1,0),(x_2,0),\ldots,(x_{i-1},0),(x_i,1),(x_{i+1},0),(x_{i+2},0)\ldots,(x_k,0)$
  Then $f(x)=\frac{(x-x_1)(x-x_2)...(x-x_{i-1})(x-x_{i+1})(x-x_{i+2})...(x-x_k)}{(x_i-x_1)(x_i-x_2)...(x_i-x_{i-1})(x_i-x_{i+1})(x_i-x_{i+2})...(x_i-x_k)}$

Lagrange Interpolation - Basis

- Consider $(1,0),(2,1),(3,0),(4,0),\ldots,(k,0)$
  Then $f(x)=\frac{(x-1)(x-3)(x-4)...(x-k)}{(2-1)(2-3)(2-4)...(2-k)}$
- Consider $(x_1,0),(x_2,0),\ldots,(x_{i-1},0),(x_i,1),(x_{i+1},0),(x_{i+2},0)\ldots,(x_k,0)$
  Then $f(x)=\frac{(x-x_1)(x-x_2)...(x-x_{i-1})(x-x_{i+1})(x-x_{i+2})...(x-x_k)}{(x_i-x_1)(x_i-x_2)...(x_i-x_{i-1})(x_i-x_{i+1})(x_i-x_{i+2})...(x_i-x_k)}$
- Consider $(x_1,0),(x_2,0),\ldots,(x_{i-1},0),(x_i,y_i),(x_{i+1},0),(x_{i+2},0)\ldots,(x_k,0)$
  Then $f(x)=y_i\frac{(x-x_1)(x-x_2)...(x-x_{i-1})(x-x_{i+1})(x-x_{i+2})...(x-x_k)}{(x_i-x_1)(x_i-x_2)...(x_i-x_{i-1})(x_i-x_{i+1})(x_i-x_{i+2})...(x_i-x_k)}$

Lagrange Interpolation - Formula

- Consider $(x_1, 0), (x_2, 0), \ldots, (x_{i-1}, 0), (x_i, y_i), (x_{i+1}, 0), (x_{i+2}, 0) \ldots, (x_k, 0)$
  Then $f(x) = y_i \frac{(x-x_1)(x-x_2)...(x-x_{i-1})(x-x_{i+1})(x-x_{i+2})...(x-x_k)}{(x_i-x_1)(x_i-x_2)...(x_i-x_{i-1})(x_i-x_{i+1})(x_i-x_{i+2})...(x_i-x_k)}$

Lagrange Interpolation - Formula

- Consider $(x_1, 0), (x_2, 0), \ldots, (x_{i-1}, 0), (x_i, y_i), (x_{i+1}, 0), (x_{i+2}, 0) \ldots, (x_k, 0)$
  Then $f(x) = y_i \frac{(x-x_1)(x-x_2)...(x-x_{i-1})(x-x_{i+1})(x-x_{i+2})...(x-x_k)}{(x_i-x_1)(x_i-x_2)...(x_i-x_{i-1})(x_i-x_{i+1})(x_i-x_{i+2})...(x_i-x_k)}$

- Consider $(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)$
  Then $f(x) = \sum_{i=1}^{k} y_i \frac{(x-x_1)(x-x_2)...(x-x_{i-1})(x-x_{i+1})(x-x_{i+2})...(x-x_k)}{(x_i-x_1)(x_i-x_2)...(x_i-x_{i-1})(x_i-x_{i+1})(x_i-x_{i+2})...(x_i-x_k)}$
  Or simply $f(x) = \sum_i y_i \prod_{j \neq i} \frac{x-x_j}{x_i-x_j}$

Lagrange Interpolation - Sum of powers

$$\sum_{k=1}^{n} k = 1 + 2 + \cdots + n = \frac{n(n+1)}{2}$$

$$\sum_{k=1}^{n} k^2 = 1^2 + 2^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6}$$

Find $\sum_{k=1}^{n} k^p = 1^p + 2^p + \cdots + n^p$
given the fact it is a degree $p+1$ polynomial of n

Lagrange Interpolation - Sum of powers

Find $\sum_{k=1}^{n} k^3 = 1^3 + 2^3 + \cdots + n^3$

$$
\begin{aligned}
0 &= 0 \\
1^3 &= 1 \\
1^3 + 2^3 &= 9 \\
1^3 + 2^3 + 3^3 &= 36 \\
1^3 + 2^3 + 3^3 + 4^3 &= 100
\end{aligned}
$$

Lagrange Interpolation - Sum of powers

Find $\sum_{k=1}^{n} k^3 = 1^3 + 2^3 + \cdots + n^3$

$$(x_i, y_i) = (0,0), (1,1), (2,9), (3,36), (4,100)$$

$$\sum_{k=1}^{n} k^3 = 1^3 + 2^3 + \cdots + n^3$$

$$= 0\frac{(x-1)(x-2)(x-3)(x-4)}{(0-1)(0-2)(0-3)(0-4)} + 1\frac{(x-0)(x-2)(x-3)(x-4)}{(1-0)(1-2)(1-3)(1-4)} +$$
$$9\frac{(x-0)(x-1)(x-3)(x-4)}{(2-0)(2-1)(2-3)(2-4)} + 36\frac{(x-0)(x-1)(x-2)(x-4)}{(3-0)(3-1)(3-2)(3-4)} +$$
$$100\frac{(x-0)(x-1)(x-2)(x-3)}{(4-0)(4-1)(4-2)(4-3)}$$

Lagrange Interpolation - Sum of powers

Find $\sum_{k=1}^{n} k^3 = 1^3 + 2^3 + \cdots + n^3$

$$(x_i, y_i) = (0,0), (1,1), (2,9), (3,36), (4,100)$$

$$\sum_{k=1}^{n} k^3 = 1^3 + 2^3 + \cdots + n^3$$

$$= \frac{x^4 - 9x^3 + 26x^2 - 24x}{-6} + 9\frac{x^4 - 8x^3 + 19x^2 - 12x}{4} +$$

$$36\frac{x^4 - 7x^3 + 14x^2 - 8x}{-6} + 100\frac{x^4 - 6x^3 + 11x^2 - 6x}{24}$$

$$= \frac{x^4}{4} + \frac{x^3}{2} + \frac{x^2}{4} = \left[\frac{n(n+1)}{2}\right]^2$$

Lagrange Interpolation - Related tasks

- HKOI `M1951`
- Codeforces `622F`

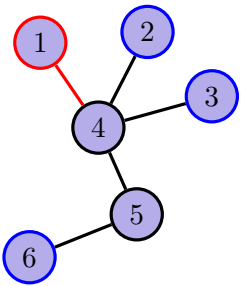
Prufer sequence - Tree to sequence

A bijection between labelled trees of n vertices and $\mathbb{Z}_n^{n-2}$ for $n = 2, 3, \ldots$

1. Remove a leaf with the smallest label until 2 vertices left
2. Append the label of the parent of each removed leaf to the resulting sequence

Prufer sequence - Tree to sequence

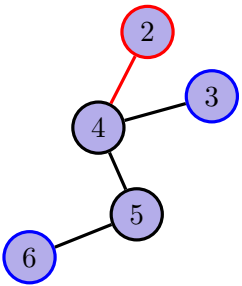
1. Remove a leaf with the smallest label until 2 vertices left
2. Append the label of the parent of each removed leaf to the resulting sequence



Sequence: (empty)

Prufer sequence - Tree to sequence

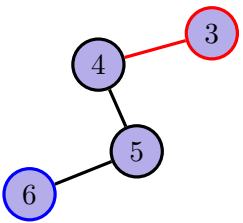
1. Remove a leaf with the smallest label until 2 vertices left
2. Append the label of the parent of each removed leaf to the resulting sequence



Sequence: 4

Prufer sequence - Tree to sequence

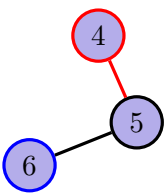
1. Remove a leaf with the smallest label until 2 vertices left
2. Append the label of the parent of each removed leaf to the resulting sequence



Sequence: $4, 4$

Prufer sequence - Tree to sequence

1. Remove a leaf with the smallest label until 2 vertices left
2. Append the label of the parent of each removed leaf to the resulting sequence

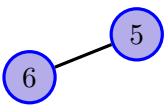


Sequence: $4, 4, 4$

Prufer sequence - Tree to sequence

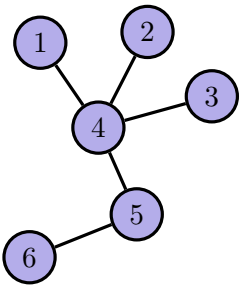
1. Remove a leaf with the smallest label until 2 vertices left
2. Append the label of the parent of each removed leaf to the resulting sequence

Sequence: $4, 4, 4, 5$



Prufer sequence - Tree to sequence

1. Remove a leaf with the smallest label until 2 vertices left
2. Append the label of the parent of each removed leaf to the resulting sequence



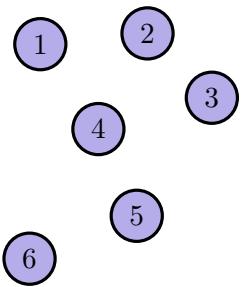
Sequence: $4, 4, 4, 5$

Prufer sequence - Sequence to tree

1. Set the missing degree of each vertices to be the number of its appearance in the sequence $+ 1$
2. Find the corresponding child of each parent in the sequence one by one which should be the vertex with missing degree 1 (for tie break, the vertex with smaller label will be chosen)
3. Build a corresponding edge for each of the child-parent pair and decrease their missing
4. After processing the whole sequence, there should be exactly two non-zero missing degrees which are both 1, build an edge between the two corresponding vertices

Prufer sequence - Sequence to tree

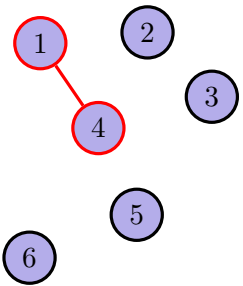
Vertex	1	2	3	4	5	6
Missing degree	1	1	1	4	2	1



Sequence: $4, 4, 4, 5$

Prufer sequence - Sequence to tree

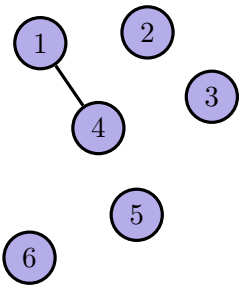
Vertex	1	2	3	4	5	6
Missing degree	0	1	1	3	2	1



Sequence: $4, 4, 4, 5$

Prufer sequence - Sequence to tree

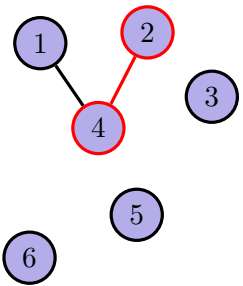
Vertex	1	2	3	4	5	6
Missing degree	0	1	1	3	2	1



Sequence: $4, 4, 4, 5$

Prufer sequence - Sequence to tree

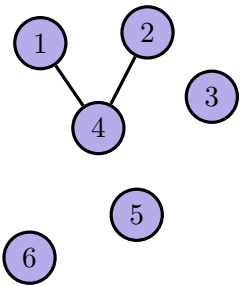
Vertex	1	2	3	4	5	6
Missing degree	0	0	1	2	2	1



Sequence: $4, 4, 4, 5$

Prufer sequence - Sequence to tree

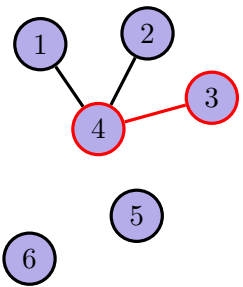
Vertex	1	2	3	4	5	6
Missing degree	0	0	1	2	2	1



Sequence: $4, 4, 4, 5$

Prufer sequence - Sequence to tree

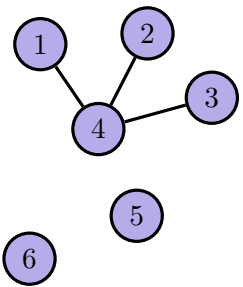
Vertex	1	2	3	4	5	6
Missing degree	0	0	0	1	2	1



Sequence: $4, 4, 4, 5$

Prufer sequence - Sequence to tree

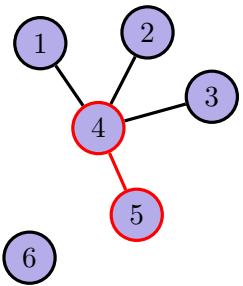
Vertex	1	2	3	4	5	6
Missing degree	0	0	0	1	2	1



Sequence: $4, 4, 4, 5$

Prufer sequence - Sequence to tree

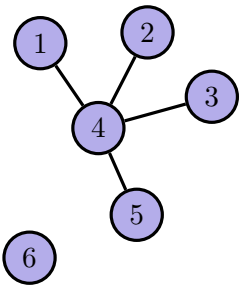
Vertex	1	2	3	4	5	6
Missing degree	0	0	0	0	1	1



Sequence: $4, 4, 4, 5$

Prufer sequence - Sequence to tree

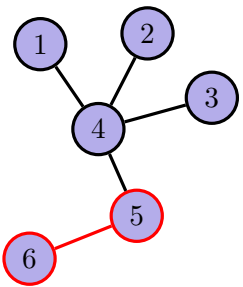
Vertex	1	2	3	4	5	6
Missing degree	0	0	0	0	1	1



Sequence: $4, 4, 4, 5$

Prufer sequence - Sequence to tree

Vertex	1	2	3	4	5	6
Missing degree	0	0	0	0	0	0



Sequence: $4, 4, 4, 5$

Prufer sequence - Cayley's Formula

- Cayley's Formula: the number of labelled trees of n vertices $= n^{n-2}$
  (also equal to the number of spanning trees
  of a complete graph of n vertices)

Prufer sequence - Cayley's Formula

- Cayley's Formula: the number of labelled trees of n vertices $= n^{n-2}$
  (also equal to the number of spanning trees of a complete graph of n vertices)
- Example: the number of labelled trees of 4 vertices $= 4^{4-2} = 4^2 = 16$

Prufer sequence - Cayley's Formula

- Cayley's Formula: the number of labelled trees of n vertices $= n^{n-2}$
  (also equal to the number of spanning trees of a complete graph of n vertices)
- Example: the number of labelled trees of 4 vertices $= 4^{4-2} = 4^2 = 16$
- Proof: there are n^{n-2} corresponding Prufer sequences

Prufer sequence - Cayley's Formula

- Cayley's Formula: the number of labelled trees of n vertices $= n^{n-2}$
  (also equal to the number of spanning trees of a complete graph of n vertices)
- Example: the number of labelled trees of 4 vertices $= 4^{4-2} = 4^2 = 16$
- Proof: there are n^{n-2} corresponding Prufer sequences
- Generalization: the number of labelled forests of n vertices and k connected components (i.e. k trees) where vertices $1, 2, \ldots, k$ are pairwise disconnected is kn^{n-k-1}

Prufer sequence - Related tasks

- Task involving counting trees
- Two-step task involving encoding trees

More topics

Most of them are not required for OI but ICPC

- Fast Fourier transform (FFT) and Number Theoretic Transform (NTT)
  - implementation of Discrete Fourier Transform (DFT)
  - multiplication of large numbers or polynomials
- Fast Walsh-Hadamard Transform (FWT)
  - a variation of the above (replacing addition by a bitwise operator in calculating the index that each pair of terms contributes to)
- Formal power series and generating function
  - generalizations of polynomials
  - useful in counting
- Linear programming and simplex method
  - optimize a linear objective function given a system of linear inequalities

More topics

Most of them are not required for OI but ICPC

- Random walk
  - a process that an object goes to one of its neighboring position randomly in each time unit
- Burnside Lemma and Polya Enumeration Theorem
  - counting the number of colourings of a cube with K colours
- Young Tableau and hook length formula
  - can be viewed as a generalization of Catalan Number
- Stirling Number
  - first kind: number of permutations of N elements and K disjoint cycles
  - second kind: number of ways to partition a set of N elements into K nonempty subsets

More topics

Most of them are not required for OI but ICPC

- Lucas's Theorem
  - calculate $C_r^n \bmod p$ with small prime p but large n
- Derangement
  - permutation without fixed points
- Permutation pattern
  - Erdos-Szekeres Theorem: HKOI J172
  - stack-sortable permutation: avoid pattern 231
- Baby-step giant-step algorithm
  - calculate discrete logarithm modular N in $O(\sqrt{N})$

More topics

Most of them are not required for OI but ICPC

- Numerical integration
  - estimate definite integral without doing calculus
- Quadratic residue
  - square number in modular sense
  - check whether a larger integer is square number with high accuracy
- Pollard's Rho Algorithm
  - find a divisor of a composite number N in $O(N^{\frac{1}{4}})$ with 50% probability
- Mobius Inversion Formula and Du's Sieve (杜教篩)
  - prefix sum of multiplicative function

More topics

Most of them are not required for OI but ICPC

- Summation on floor or ceiling function
  - calculate $\sum_i \lfloor \frac{pi}{q} \rfloor$ efficiently
- Continued fractions and Stern-Brocot Tree
  - best rational approximation: GCJ 2019 round 2 C
- Berlekamp-Massey Algorithm
  - find linear recurrence given some starting terms
- Miller-Rabin Primality Test
  - polylogarithmic time complexity