Miscellaneous math

Fast pow, Fibonacci, tortoise and hare

beCP Training

OLYMPIADE BELGE D'INFORMATIQUE
BELGISCHE INFORMATICA-OLYMPIADE

April 16, 2016

Table of Contents

Powers

Definition:

- Chain multiplication
- "n-th power of b"
- b is the base, n is the exponent
- $b^n = \underbrace{b \times \cdots \times b}_{n \text{ times}}$

Examples:

- $3^0 = 1$ (by definition)
- $3^1 = 3$
- $3^2 = 3 \times 3 = 9$ (square)
- $3^3 = 3 \times 3 \times 3 = 27$ (cube)

Power computation: linear

Problem: compute the power the n-th power of b, for given b and n.

Solution 1: Simple loop

```
int nthPower(int b, int n)
{
    int power = 1;
    for (int i = 0; i < n; i++)
        power *= b;
    return power;
}
```

Complexity: $O(n)$

Power computation: logarithmic (1)

Can we do it faster? Yes, because associativity!

For example, to compute 3^{10}, we can compute 3^5 then square it:

- $3^2 = 3 \times 3 = 9$
- $3^5 = 3^2 \times 3^2 \times 3 = 9 \times 9 \times 3 = 243$
- $3^{10} = 3^5 \times 3^5 = 243 \times 243 = 59049$

Only 4 multiplications instead of 9.

Power computation: logarithmic (2)

Solution 2: Recursive function

```
int nthPower(int b, int n)
{
    // Initial case
    if (n == 0)
        return 1;

    // Recursive case
    int power = nthPower(b, n/2);
    power *= power;
    if (n % 2 == 1)
        power *= b;
    return power;
}
```

We divide n by 2 on every call $\Rightarrow O(\log n)$

Fast pow: usage

When to use it:

- When linear time is too slow
- Typically when computing a number of possibilities

Limits:

- Exponent $\leq 10^{18}$ if using `long long` (or more!)
- Many powers with the same base $\Rightarrow$ store in an array
- Be careful with overflows! Often, the statement asks for the result *modulo* some number.

Table of Contents

Matrices

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \end{pmatrix} = \begin{pmatrix} 0 & 4 & 1 \\ 2 & 0 & 0 \\ 1 & 1 & 2 \\ 8 & 0 & 1 \end{pmatrix}$$

- 4×3 matrix = table with 4 lines and 3 columns
- a_{ij} = a[i-1][j-1] = element at line i, column j
- Indexing usually starts at 1 (what a shame)

Matrix product

$$\begin{pmatrix} 0 & 4 & 1 \\ 2 & 0 & 0 \\ 1 & 1 & 2 \\ 8 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 2 & 1 \\ 0 & 3 \end{pmatrix} = \begin{pmatrix} 8 & 7 \\ 2 & 0 \\ 3 & 7 \\ 8 & 3 \end{pmatrix}$$

- Take line on left, column on right
- Add the products (they must have the same length!)
- $0 \times 1 + 4 \times 2 + 1 \times 0 = 8$ (top left element)
- Dimensions: $n \times m \ \cdot \ m \times p \ = \ n \times p$
- Associative: $(AB)C = A(BC)$

Matrix power

$$A^2 = \begin{pmatrix} 0 & 4 & 1 \\ 2 & 0 & 0 \\ 1 & 1 & 2 \end{pmatrix} \begin{pmatrix} 0 & 4 & 1 \\ 2 & 0 & 0 \\ 1 & 1 & 2 \end{pmatrix} = \begin{pmatrix} 9 & 1 & 2 \\ 0 & 8 & 2 \\ 4 & 6 & 5 \end{pmatrix}$$

- Only works on square matrices
- Associative, so we can use fast pow!
- Complexity for A^n: cost of product $\times O(\log n)$
- With $m \times m$ matrix: $O(m^3 \log n)$

Table of Contents

Definition of Fibonacci

$$0,\ 1,\ 1,\ 2,\ 3,\ 5,\ 8,\ 13,\ 21,\ 34,\ 55, \ldots$$

$$F_0 = 0,\ F_1 = 1,\ F_2 = 1, \ldots$$

- Modelling rabbit reproduction (terribly)
- At 2 months old, rabbits start having 1 child every month
- At month 1, we introduce one newborn rabbit
- F_n = population at month n
- $F_n = F_{n-1} + F_{n-2}$
- Example: $F_2 = F_1 + F_0,\ \ F_3 = F_2 + F_1,\ \ldots$

Fibonacci as a matrix product

- We want to compute Fibonacci as a matrix product
- We always need to know at least two numbers
- Which square matrix to choose?

$$\begin{pmatrix} ? & ? \\ ? & ? \end{pmatrix} \begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix} = \begin{pmatrix} F_{n+1} \\ F_{n+2} \end{pmatrix}$$

Fibonacci as a matrix product

- We want to compute Fibonacci as a matrix product
- We always need to know at least two numbers
- Which square matrix to choose?

$$\begin{pmatrix} 0 & 1 \\ ? & ? \end{pmatrix} \begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix} = \begin{pmatrix} F_{n+1} \\ F_{n+2} \end{pmatrix}$$

Fibonacci as a matrix product

- We want to compute Fibonacci as a matrix product
- We always need to know at least two numbers
- Which square matrix to choose?

$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix} = \begin{pmatrix} F_{n+1} \\ F_{n+2} \end{pmatrix}$$

Logarithmic Fibonacci

From the formula, we deduce:

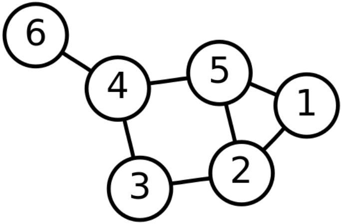
$$\begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}$$

- So we can compute the power in $O(\log n)$
- And then get F_n immediately
- Because of overflows, we often need the result *modulo* some number

Table of Contents

Adjacency matrix

- Static two-dimensional array: `int adj[MAXN][MAXN]`
- adj[i][j] $==$ true if edge $i \to j$
- adj_{ij} $=$ number of paths of length 1 from i to j



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

Number of paths of length 2

- Look at all possible intermediate nodes k
- $\text{num\_path2}_{ij} = \text{sum}(\text{adj}_{ik} \times \text{adj}_{kj})$
- adj_{ik} is line i and adj_{kj} is column j

Example: number of paths from 2 to 4:

$$\begin{pmatrix} 0 & 1 & 0 & \mathbf{0} & 1 & 0 \\ \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ 0 & 1 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 1 & \mathbf{0} & 1 & 1 \\ 1 & 1 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & 0 & 0 \end{pmatrix}^2 = \begin{pmatrix} 2 & 1 & 1 & 1 & 1 & 0 \\ 1 & 3 & 0 & \mathbf{2} & 1 & 0 \\ 1 & 0 & 2 & 0 & 2 & 1 \\ 1 & 2 & 0 & 3 & 0 & 0 \\ 1 & 1 & 2 & 0 & 3 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

This is just matrix multiplication: $\text{num\_path2} = \text{adj}^2$

Number of paths of length 3

- We look at all the intermediate nodes
- $\text{num\_path3}_{ij} = \text{sum}(\text{num\_path2}_{ik} \times \text{adj}_{kj})$
- num_path2_{ik} is line i and adj_{kj} is column j

Example: number of paths from 1 to 4

$$\underbrace{\begin{pmatrix} \mathbf{2} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{0} \\ 1 & 3 & 0 & 2 & 1 & 0 \\ 1 & 0 & 2 & 0 & 2 & 1 \\ 1 & 2 & 0 & 3 & 0 & 0 \\ 1 & 1 & 2 & 0 & 3 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}}_{\text{num\_path2}} \underbrace{\begin{pmatrix} 0 & 1 & 0 & \mathbf{0} & 1 & 0 \\ 1 & 0 & 1 & \mathbf{0} & 1 & 0 \\ 0 & 1 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 1 & \mathbf{0} & 1 & 1 \\ 1 & 1 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & 0 & 0 \end{pmatrix}}_{\text{adj}} \rightarrow 2$$

So $\text{num\_path3} = \text{num\_path2} \times \text{adj} = \text{adj}^2 \times \text{adj} = \text{adj}^3$

Number of paths of any length

- In general, the number of paths of length n can be computed with adj^n
- So for V nodes, we can get it in $O(V^3 \log n)$

Remarks:

- This also works with directed graphs (one-way edges)
- This also works with multiple edges between two nodes (put the number in the matrix)
- If you only want to know if there *is a path or not*, you can do it in $O(V+E)$ for one source, or $O(V(V+E))$ for all pairs. (Fun exercise!)

Table of Contents

Cycle finding problem

- Finite set of states S, for example $\{1, \ldots, n\}$
- Function $f : S \to S$ of transitions $x \mapsto f(x)$
- Starting value: $x_0 \in S$

What is the first value repeated in this sequence?

$$x_0,\ f(x_0),\ f(f(x_0)),\ f(f(f(x_0))), \ldots$$

Cycle finding example

Table of f:

x	1	2	3	4	5	6	7	8	9
$f(x)$	8	6	1	7	1	4	5	5	2

Sequence for $x_0 = 9$:

$$\underbrace{9,\ 2,\ 6,\ 4,\ 7}_{\text{tail}},\ \underbrace{5,\ 1,\ 8}_{\text{cycle}},\ \underbrace{5,\ 1,\ 8}_{\text{cycle}},\ 5,\ 1, \ldots$$

Problem: find the tail size and the cycle size

Cycle finding with map

```
pair<int,int> findCycle(int x)
{
    map<int,int> pos;

    // While x is a new value
    int i;
    for (i = 0; pos.find(x) == pos.end(); i++)
    {
        pos[x] = i; // Remember the position
        x = f(x);   // Move one step
    }
    return make_pair(pos[x], i - pos[x]);
}
```

- Time: $O(n \log n)$, or $O(n)$ with hashmap
- Space: $O(n)$, that's a lot...

Tortoise and hare 1: find match

Keep two pointers: tortoise (slow) and hare ($2\times$ faster)

Iterate until match:

9	2	6	4	7	5	**1**	8	5	1	8	5	**1**	8	5
	T	H												
		T		H										
			T			H								
				T				H						
					T					H				
						T						H		

The gap is a multiple of the cycle length

Tortoise and hare 2: find tail

Hare jumps to beginning, then they move at the same speed (the hare is tired by the jump)

Iterate until match:

9	2	6	4	7	**5**	1	8	5	1	8	**5**	1	8	5
H						T								
	H						T							
		H						T						
			H						T					
				H						T				
					H						T			

The number of steps is the tail length

Tortoise and hare 3: find cycle

Hare stops (*really* tired) and tortoise continues

Iterate until next match:

9	2	6	4	7	**5**	1	8	5	1	8	5	1	8	**5**
					H						T			
					H							T		
					H								T	
					H									T

The number of steps is the cycle length

Tortoise and hare: implementation

```
pair<int,int> findCycle(int x0)
{
    int t = x0, h = x0, tail = 0, cycle = 0;
    // Part 1: find a match
    do { t = f(t); h = f(f(h)); } while (t != h);
    // Part 2: find tail
    h = x0; // Rabbit jump
    while (t != h) { t = f(t); h = f(h); tail++; }
    // Part 3: find cycle
    do { t = f(t); cycle++; } while (t != h);
    return make_pair(tail, cycle);
}
```