Queue

First-in-First-out data structure

Nattee Niparnan

Intro

- Just like a normal queue in real life.
  - First-in-First-out data structure
  - One way in (back of queue), one way out (front of queue)
  - push = add data to the back of the queue
  - pop = remove data from the head of the queue

Front

Back

```
push("A")
push("B")
push("C")
pop()
push("X")
pop()
pop()
```

Basic

```cpp
#include <iostream>
#include <queue>
#include <vector>

using namespace std;

int main() {
    queue<int> q;
    q.push(1);
    q.push(2);
    q.push(3);
    while (q.empty() == false) {
        cout << q.front() << endl;
        q.pop();
    }
    cout << "-- example 2 --" << endl;
    queue<vector<int>> q2;
    vector<int> v1 = {1,2,3};
    vector<int> v2 = {99,88,-1};
    q2.push( v1 );
    q2.push( v2 );
    cout << q2.back()[1] << endl;
    cout << q2.front().size() << endl;
    auto x = q2.front();
    q2.pop();
    cout << x[0] << endl;

}
```

```
size_t      q.size()
bool        q.empty()
void        q.push(T data)
void        q.pop()
T           q.front()
T           q.back()
```

Limitation

- Same limitation as stack
  - No iterator
    - No begin(), end()
    - Can only access front and back of the queue
    - If we wish to access all members, we have to pop it all
  - Do not call front(), back(), pop() when the queue is empty

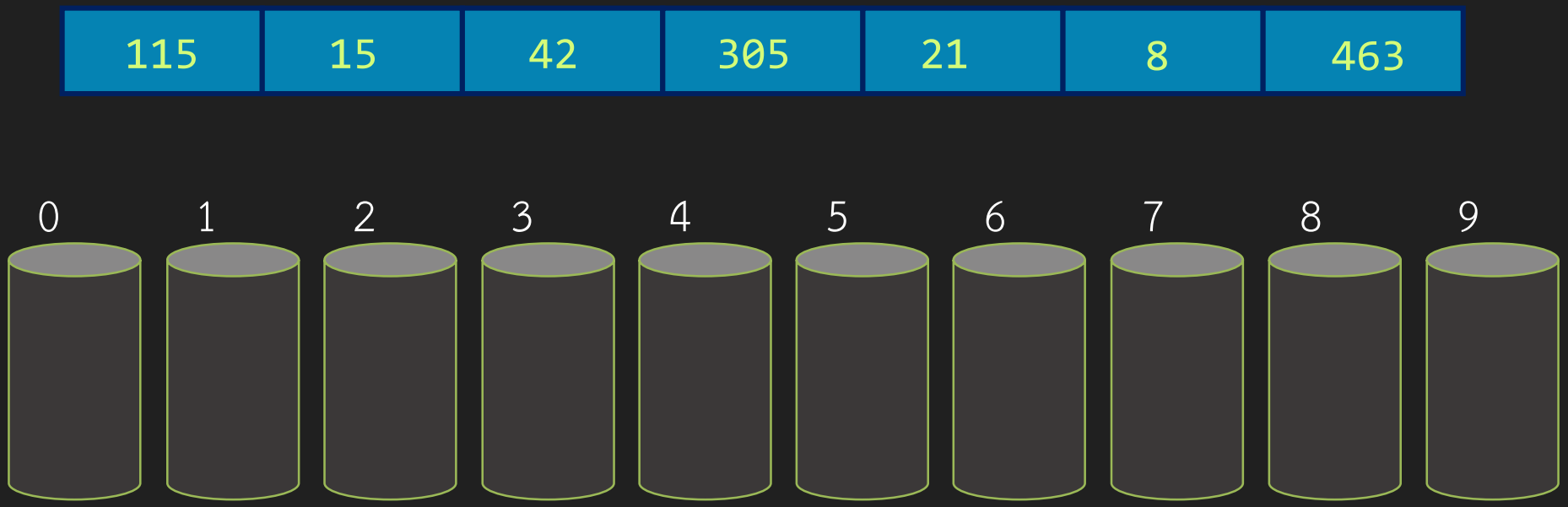
Radix Sort

Queue Application: Fast sorting with no comparison

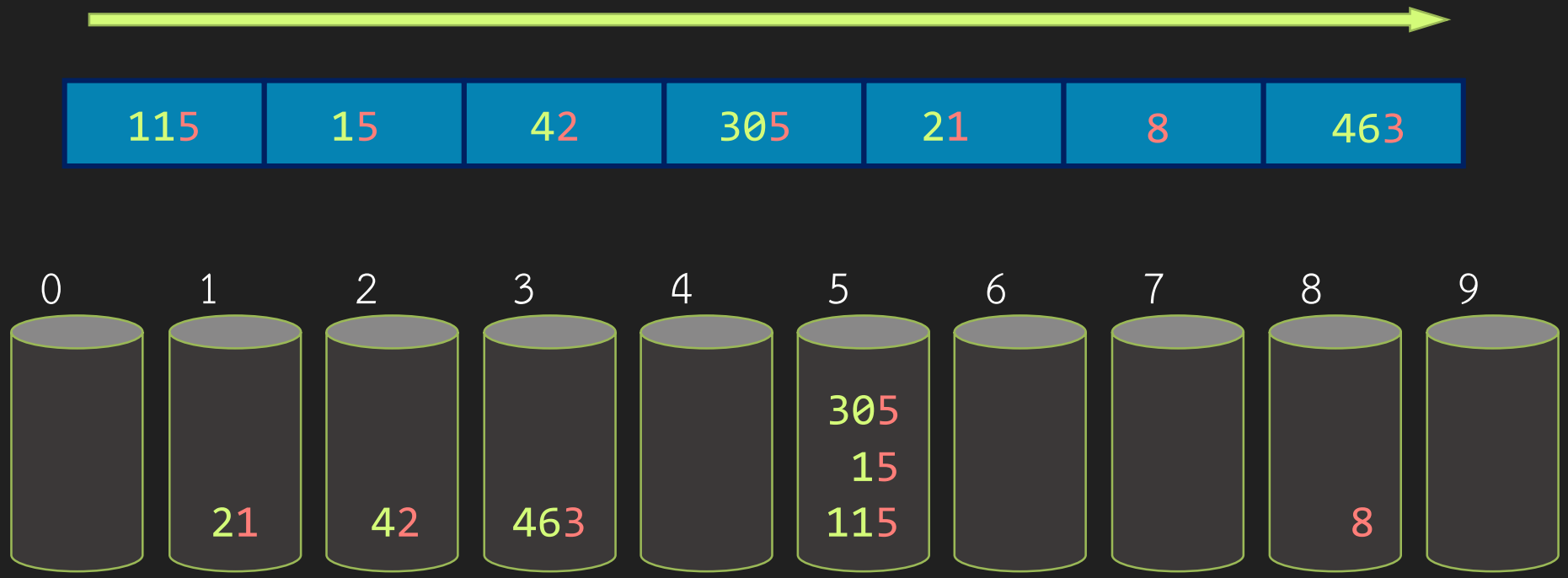
Overview

- Put all data in an array
- For each digit X, from LSD to MSD
  - PUT TO QUEUE step: Sort by digit X by putting all of data from the array into B queues
    - B is the base of the number
    - For example, for a base 10 number, we will have Queue[0] to Queue[9]
    - Put into the queue labelled with that digit
  - GET FROM QUEUE step: Start from queue 0 to queue B-1, remove data from the queue and put back to the array

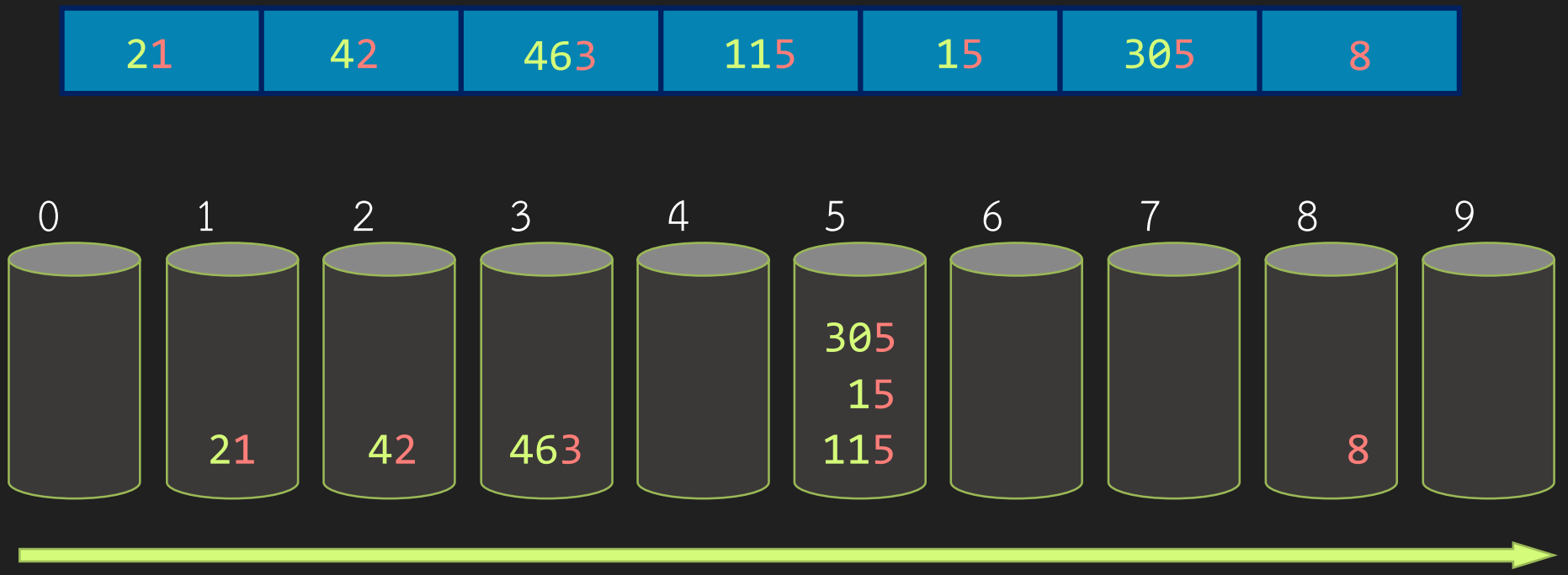
Example:



Example: Round 1 (digit 0), to queue



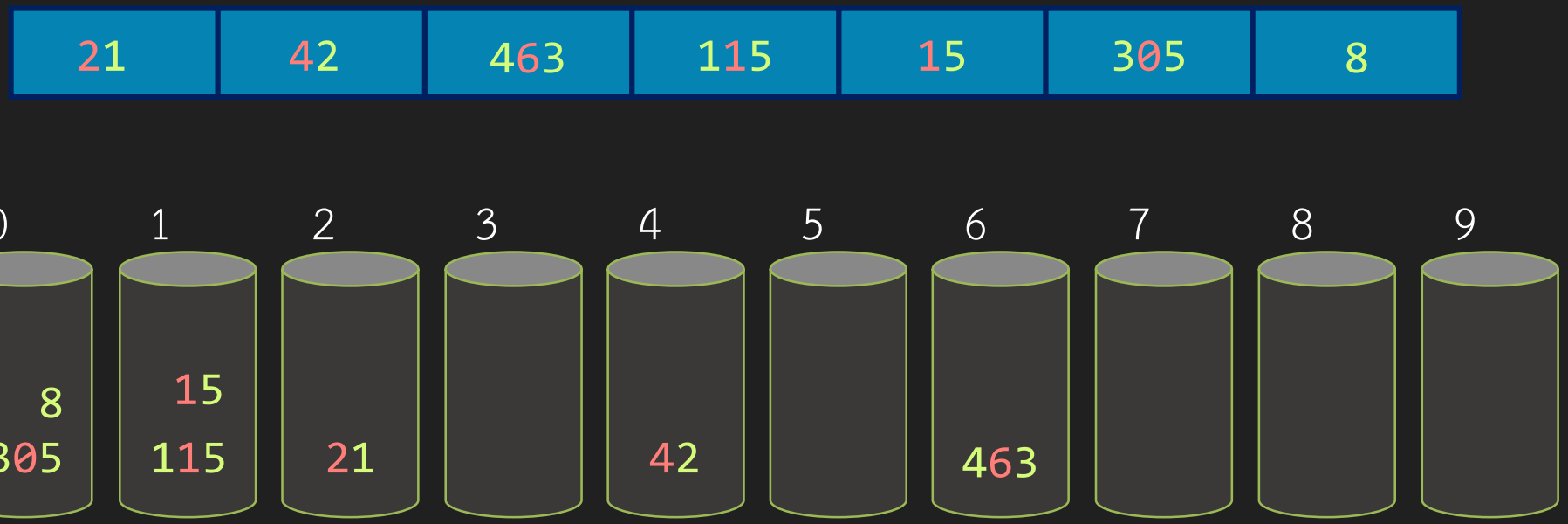
Example: Round 1 (digit 0), from queue



Data is now sorted by the last digits,

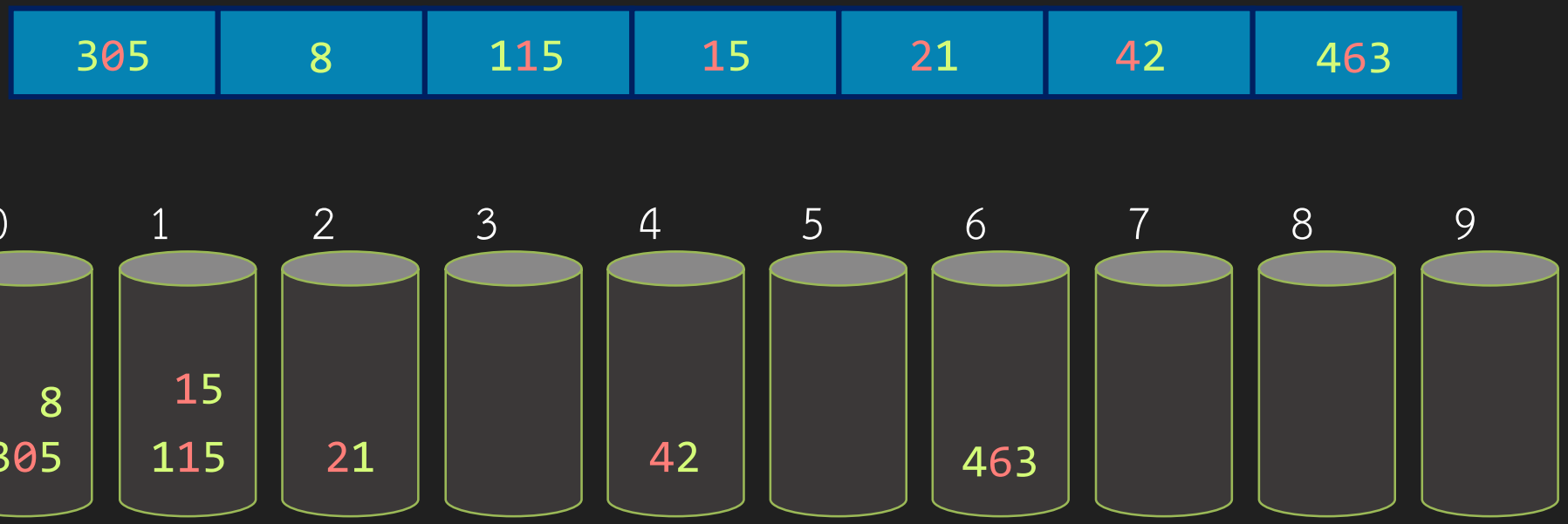
because we pop from queue 0 to 9

Example: Round 2 (digit 1), to queue



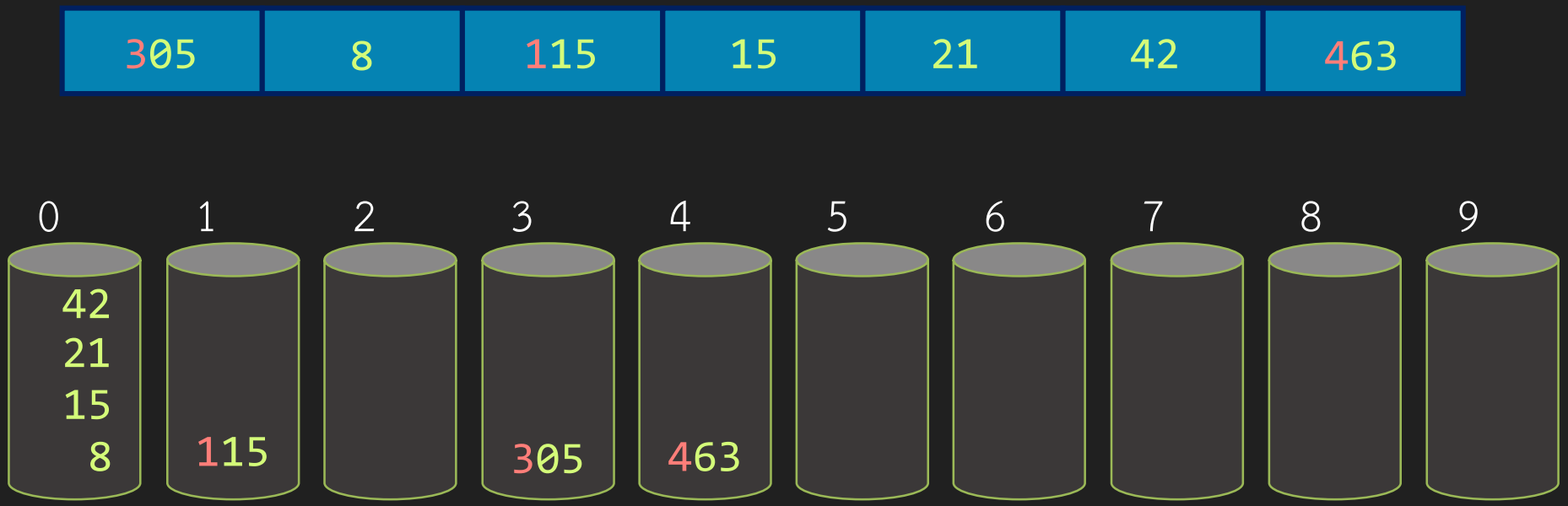
In each queue, the data is sorted by the last digit, because we goes from left to right in the array which is already sorted by the last digit

Example: Round 2 (digit 1), from queue



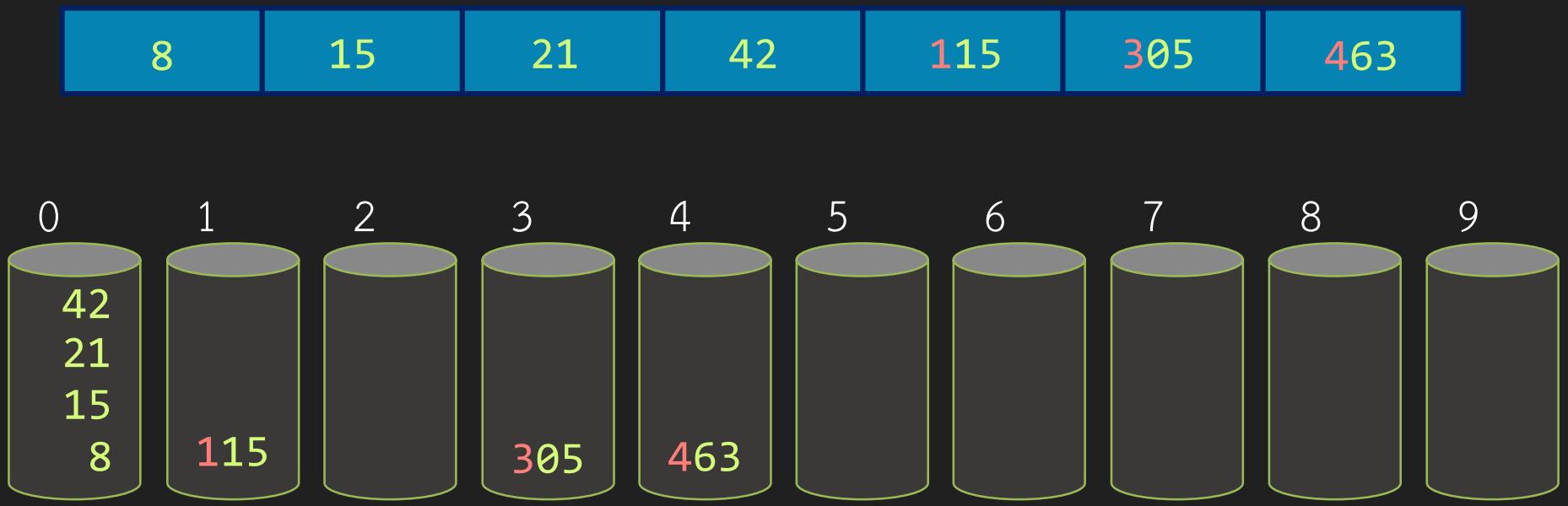
Data is now sorted by last two digits, because we goes from queue 0 to 9, which is grouped by digit 1 and the data in each queue is sorted by the last digit.

Example: Round 3 (digit 2), to queue



In each queue, the data is sorted by the last two digits, because we goes from left to right in the array which is already sorted by the last two digits

Example: Round 3 (digit 2), from queue



Data is now sorted by all digits, because we goes from queue 0 to 9, which is grouped by digit 2 and the data in each queue is sorted by the last two digits.

Code

```
#define base 10

int getDigit(int v, int k) {
    // return the kth digit of v (MSD is digit 0)
    int i;
    for (i=0; i<k; i++) v /= base;
    return v % base;
}

// d = number of digits
void radixSort(vector<int> &data,int d) {
    queue<int> q[base];
    for (int k=0; k<d; k++) {
        for (auto &x : data)
            q[getDigit(x,k)].push(x);
        for (int i=0,j=0; i<base; i++)
            while(!q[i].empty()) {
                data[j++] = q[i].front(); q[i].pop();
            }
    }
}
```

Breadth First Search

Queue Application: Gotta generate ’em all!

The Problem

- Given a positive integer X
- Start with a number 1, find a sequence of arithmetics operations, either "* 3", or "/ 2" that makes 1 into X
  - the / 2 is integer division, e.g., 5 / 3 = 1 (not 1.6666)
  - The sequence must be as short as possible
- Example
  - 10 = 1 * 3 * 3 * 3 * 3 / 2 / 2 / 2
  - 31 = 1 * 3 * 3 * 3 * 3 * 3 / 2 / 2 / 2 / 2 / 2 * 3 * 3 / 2

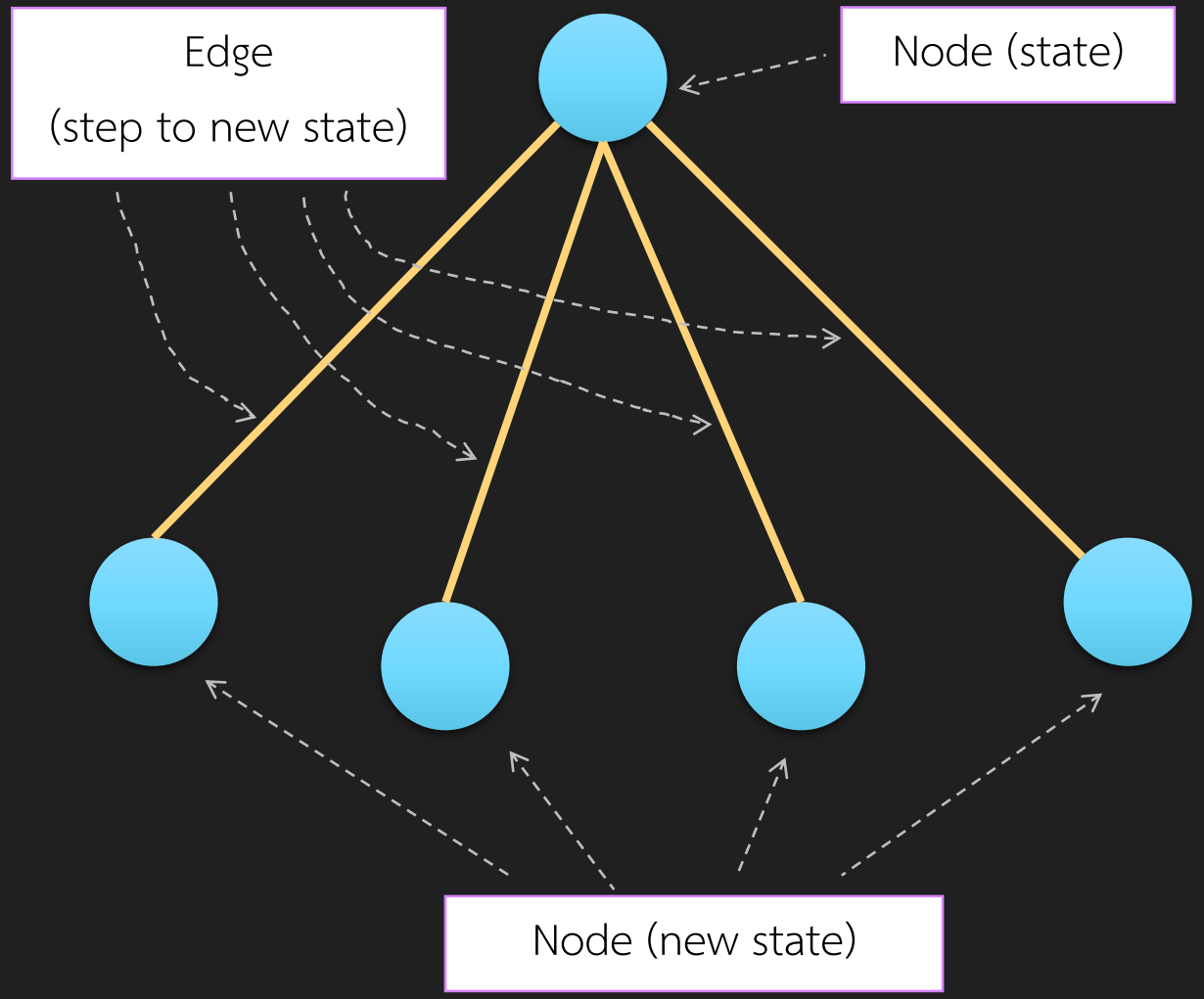
The Idea

- Generate all possible sequences
  - Start with length 1, 2, 3, … until we find one that gives X
- This is called an exhaustive search algorithm
  - Systematically enumerate all possible somethings

Tree Structure

- A structure to illustrates search algorithm
- Divide into steps
  - Start with initial solution
  - For each possible outcome (called a state) of each step, genenate all proper possible next step
    - Also, check if the current step is what we need
- Written as a diagram of node and edge

Enumerate

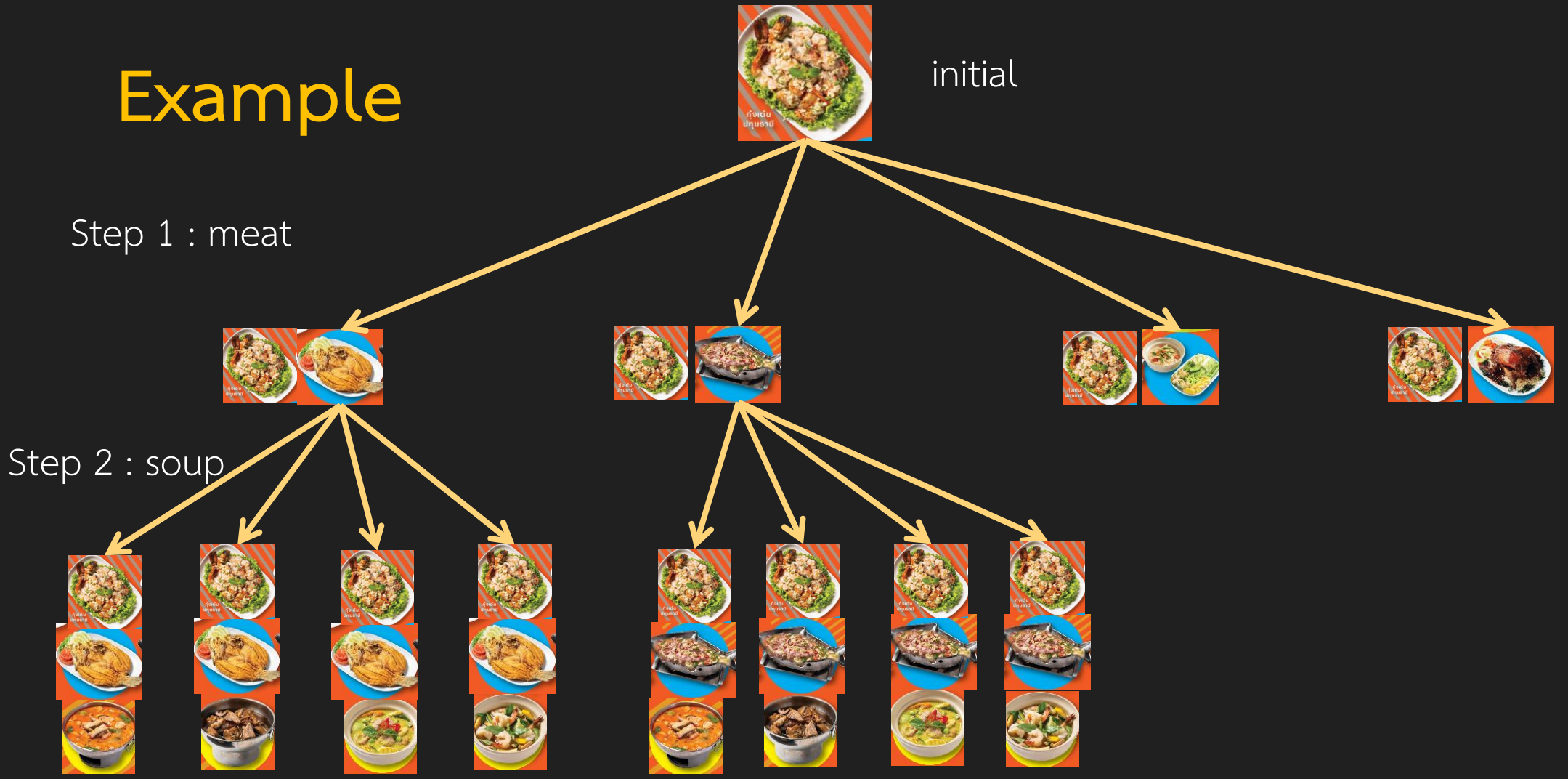


- Write a diagram, each state is a node (drawn as a circle)
- Enumerate all possible next steps of each state as edges (drawn as a line)
  - Doing each step will result in a new state

Example

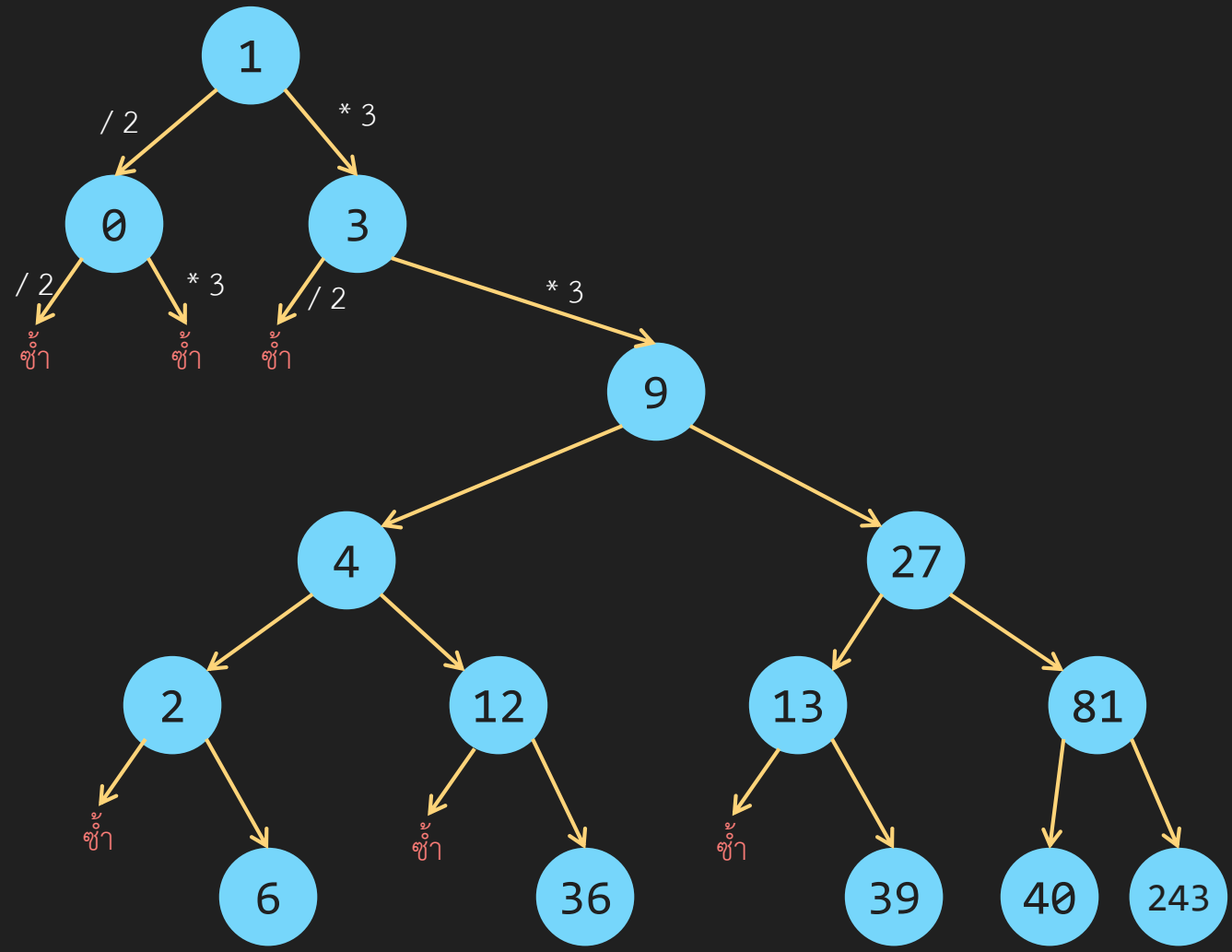
- Enumerate all possible meals from this promotion
  - There are 3 steps: Meat, Soup, Veggie
  - Each step we have to pick one
- Initial solution = starting order
- Each step, pick one of the choice and put into order

Example



Back to our problem

- Start with 1
- Each step is either * 3 or / 2
- Issue: might get repeated number
  - Solution: if we have found it, do not generate new step



Code

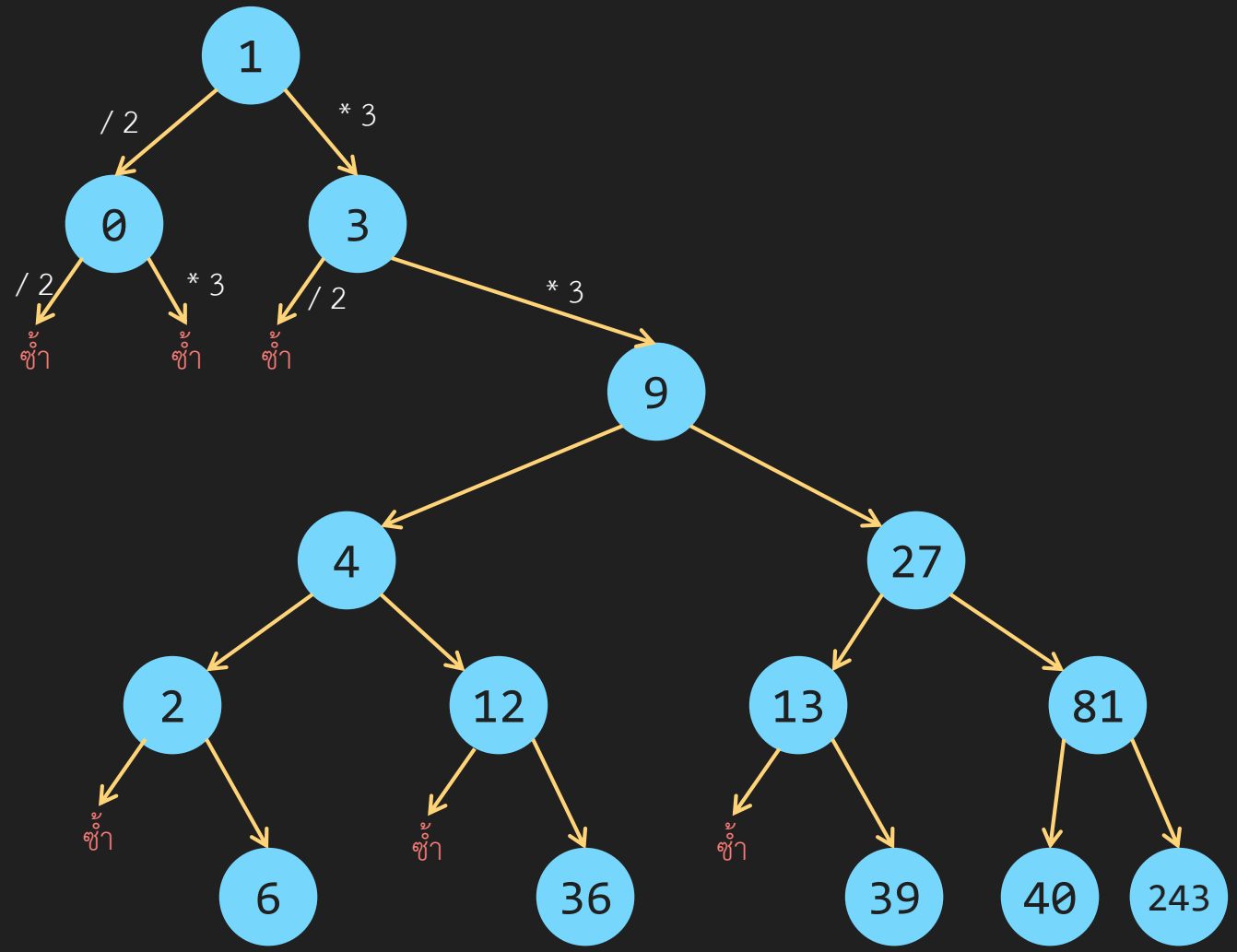
```
void m3d2(int target) {
    map<int, int> prev;
    queue<int> q;
    int v = 1;
    q.push(1); prev[1] = -1;
    while( !q.empty() ) {
        v = q.front(); q.pop();
        if (v == target) break;
        int v2 = v/2;
        int v3 = v*3;
        if (prev[v2] == 0) {q.push(v2); prev[v2] = v;}
        if (prev[v3] == 0) {q.push(v3); prev[v3] = v;}
    }
    if (v == target) showSolution(v, prev);
}
```

- Queue makes ordering of how we pick a state to enumerate
- From top to bottom and left to right

Display Solution

- Trace back “prev”

X	prev[x]
0	1
1	-1
2	4
3	1
4	9
6	2
9	3
12	4
13	27
27	9
36	12
39	13
40	81



showSolution

```
void showSolution(int v, map<int,int>& prev) {
    string out = "";
    while(prev[v] != -1) {
        if (prev[v] * 3 == v) {
            out = "x3" + out;
        } else {
            out = "/2" + out;
        }
        v = prev[v];
    }
    out = "1" + out;
    cout << out << endl;
}
```