Data Structures

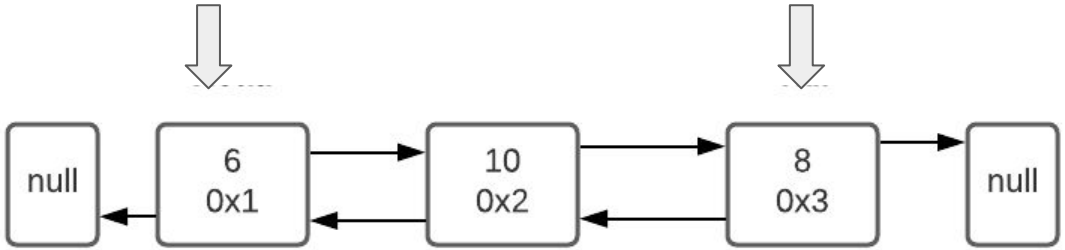
Drawings

Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

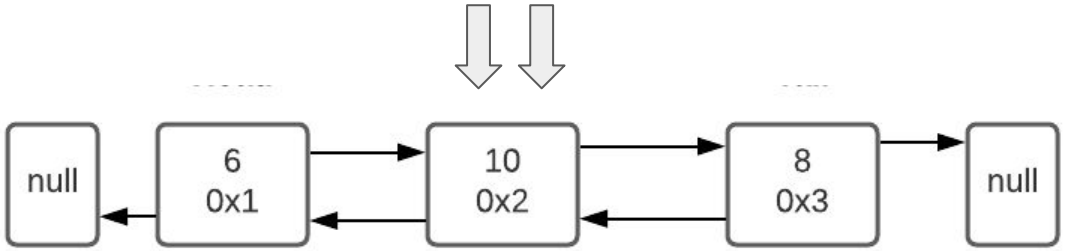
Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

Problem #1: Find the middle using DLL

- Assume an odd number of nodes
- Let’s start from head and tail and move one step
- We must come to a point where both are at the same node: forward=backward

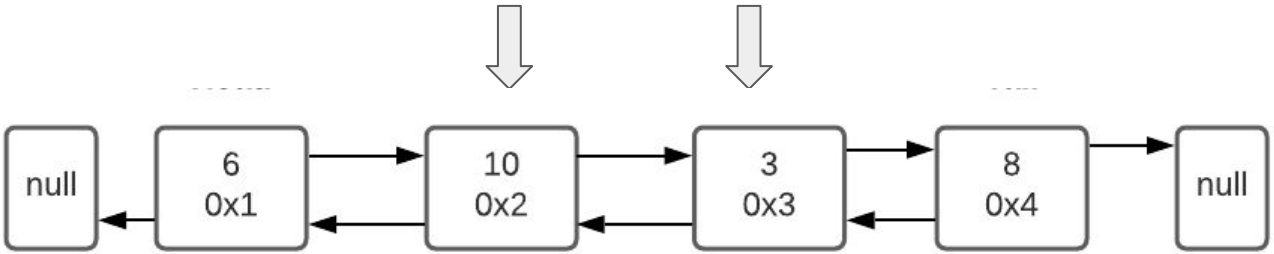


Problem #1: Find the middle using DLL



Problem #1: Find the middle using DLL

- For the even number of nodes, the 2 nodes will be neighbours!
- For odd or even cases, move the head and tail until either:
  - They are equal (odd)
  - They are neighbours (even)
- h != t and h.next != t

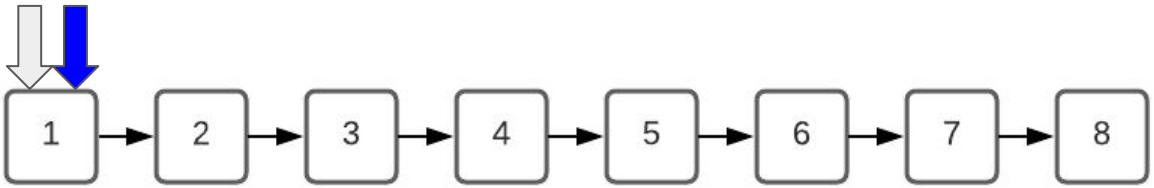


Problem #1: Find the middle using **SLL**

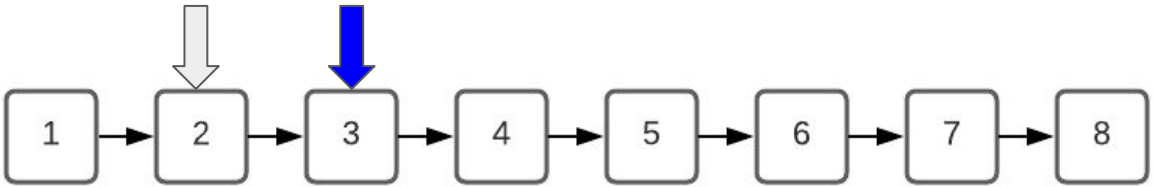
- Only using the 'next' node is an interesting idea, but it's hard to get by yourself
- Use 2 nodes:
  - The first (slow) moves normally step by step
  - The second (fast) jump 2 steps each time!
- For example, if a list has 10 elements:
  - When the 'slow' pointer is in the middle (5), the 'fast' one will be double that - at the end! (10)
- From that, we know we found the middle.

Problem #1: Find the middle using SLL

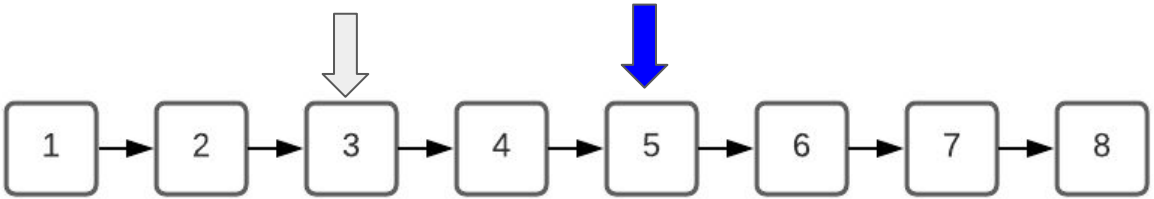
- Assume Blue is the fast pointer
  - Slow moves: 1, 2, 3, 4, 5, 6, 7, 8
  - Fast moves: 1, 3, 5, 7, null
  - Once fast is done, slow MUST be at the middle



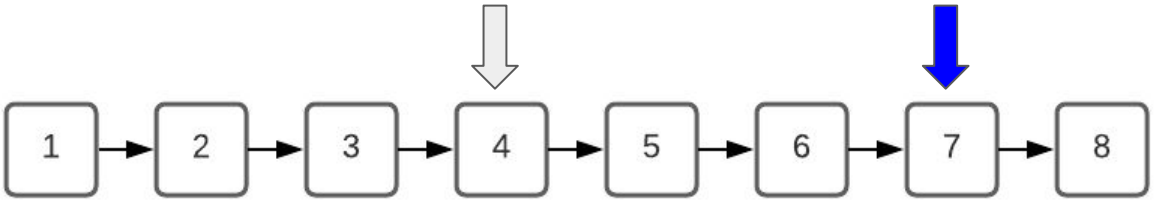
Problem #1: Find the middle using SLL



Problem #1: Find the middle using SLL

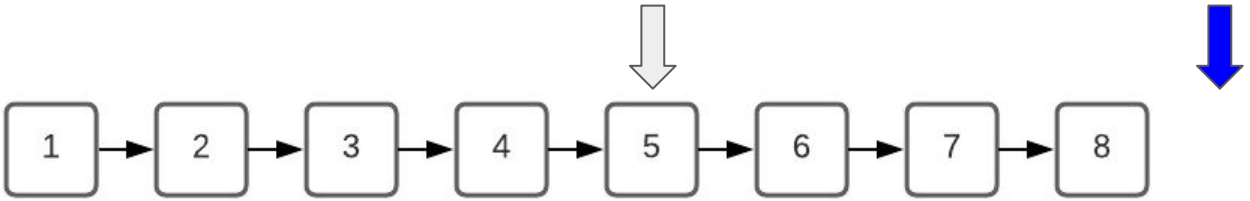


Problem #1: Find the middle using SLL



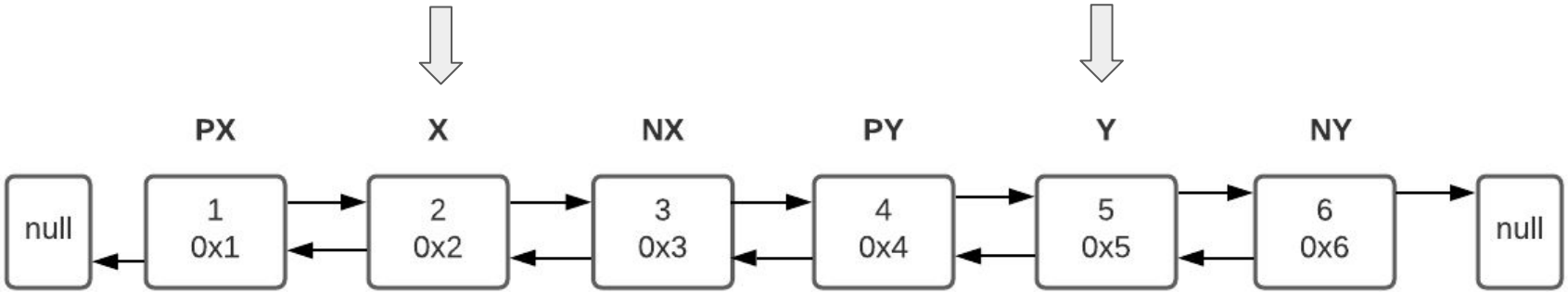
Problem #1: Find the middle using SLL

- As we need the 2nd middle, in both even/odd we see we are at right location
- *The idea is based on the tortoise and hare algorithm [no need to check]*
  - You may meet this again



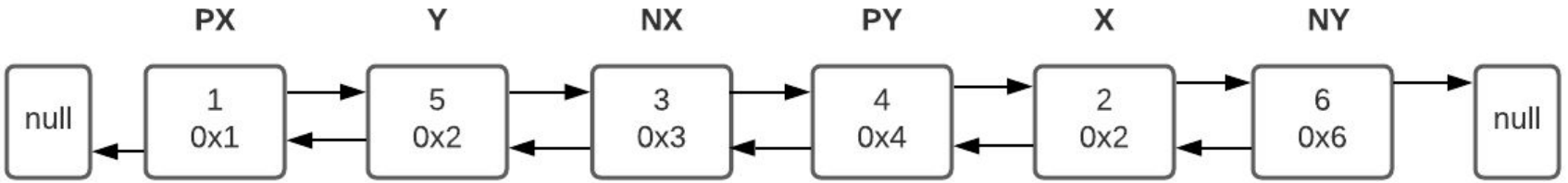
Problem #2: Swap forward with backward

- Assume k = 2. Let’s say our nodes are X and Y
- Let’s say previous and next of X are PX and NX
  - Same for Y: PY and NY
- Take copies from all
- Draw the before and after. This helps us to relink them trivially



Problem #2: Swap forward with backward

- Clearly we need to link the following pairs: (PX, Y), (Y, NX), (PY, X), (X, NY)
- Careful with cases:
  - Both nodes are the same. This only happens for an odd length list
  - First node is after the last node. E.g. for K = 5
  - They are consecutive (neighbours). E.g. for K = 3
- It's trivial to detect these using the 'length' variable. It takes a little more effort without it



Problem #3: Reverse list nodes

- With SLL, reverse is easy. With a DLL, a little more caution is required
- Move left to right. Reverse the current 2 nodes, then move:
  - Assume the list has nodes: A, B, C, D, E, F
  - Start from the beginning, with 2 consecutive nodes: A and B
  - Take copies of the next 2 nodes (B, C)
  - Link(B, A) now they are reversed
    - B, A C, D, E, F
  - Move one step using the copies of B and C
  - In the next step it will be
  - C, B, A D, E, F
  - In end, handle proper head & tail

Problem #4: Merge lists

- Start from a None node (we can do that in a DLL, but not in a SLL)
- In each step, see which one has the smaller value. Pick from them
  - Keep doing this for as long as both sequences are not finished!
- After that, one of them remains. Simply connect the remainder of whatever sequence remains with all of the sequence that has already been constructed!
- Maintain correct data integrity

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."