

Lecture 4

Linked List

Our Roadmap

- ◆ Linked List Definition
- ◆ Linked List Operators
- ◆ Illustration Example

Representing a Sequence of Data

- ◆ An ordered collection of items (position matters)
 - ◆ Array, lists, stacks, and queues
- ◆ What did you study before? Array!
- ◆ Advantages of using an array
 - ◆ Easy and efficient access to any item in the sequence
 - ◆ `item[i]`: return the i-th element in array item
 - ◆ Every item can be accessed in constant time
 - ◆ This feature of arrays is known as “random access”
 - ◆ Very compact (in terms of memory)
- ◆ Disadvantages of using an array ?

Disadvantages of an Array

- ◆ Have to specify an initial array size
- ◆ Resize an array is possible, but not so easy
- ◆ Difficult to insert/delete elements at arbitrary positions
 - ◆ Delete **10** in array A, time complexity?

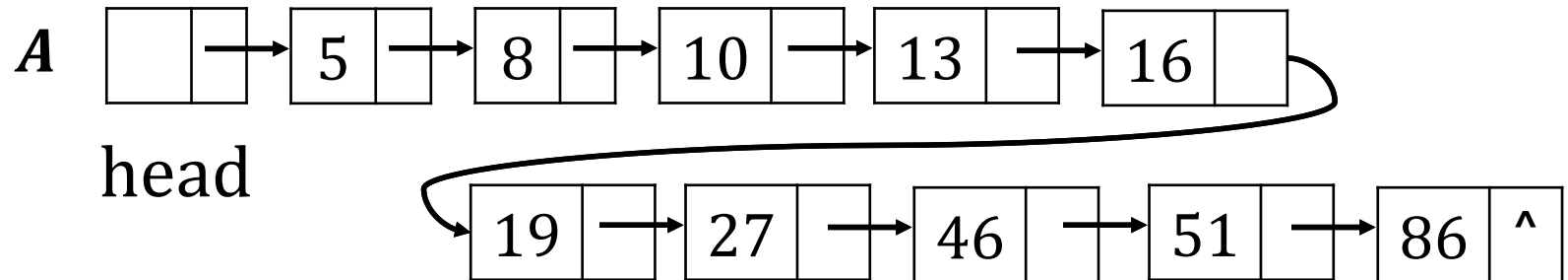
A	5	8	10	13	16	19	27	46	51	86
----------	---	---	----	----	----	----	----	----	----	----

A	5	8		13	16	19	27	46	51	86
----------	---	---	--	----	----	----	----	----	----	----

A	5	8	13	16	19	27	46	51	86	
----------	---	---	----	----	----	----	----	----	----	--

A Linked List

- ◆ Alternative Representation of a sequence. Example:



- ◆ A linked list stores a sequence of elements in separate nodes
- ◆ Each node contains: a single item, a “link” to the node containing the next item:

13	→
----	---
- ◆ The last node in the linked list has a link value of “NULL”:

86	^
----	---
- ◆ The linked list as a whole is represented by a variable that hold a reference to the first node (e.g., *A*)

Array vs. Linked List in Memory

- ◆ In an array, the elements occupy consecutive memory locations:

5	8	10	13	16	19	27	46	51	86
0x100	0x104	0x108	0x112	0x116	0x120	0x124	0x128	0x132	0x136

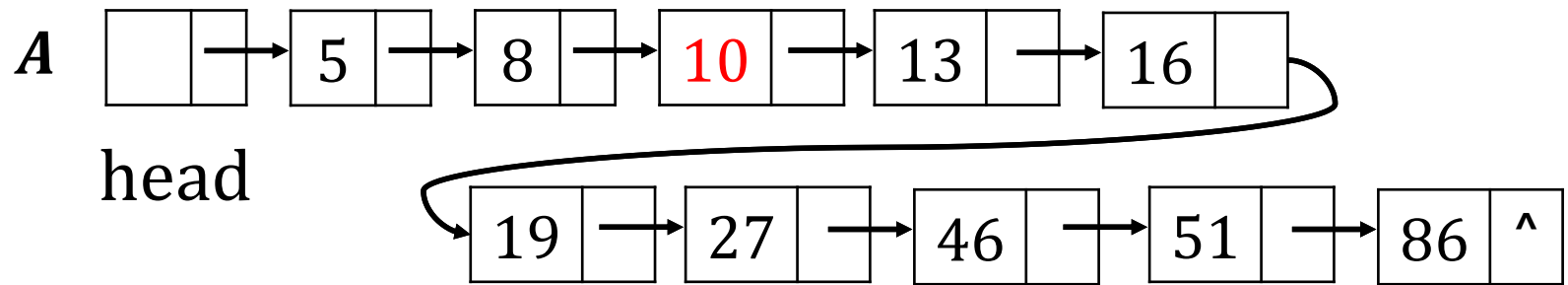
- ◆ In linked list, each node is a distinct object. The nodes do NOT have to be next to each other in memory. That's why we need the links to get from one node to the next.

0x100	8
	0x480
...	...
0x240	13
	0x640
...	...
	5
	0x100
...	...
0x480	10
	0x240
...	...
0x640	16
	0x800
.....	...

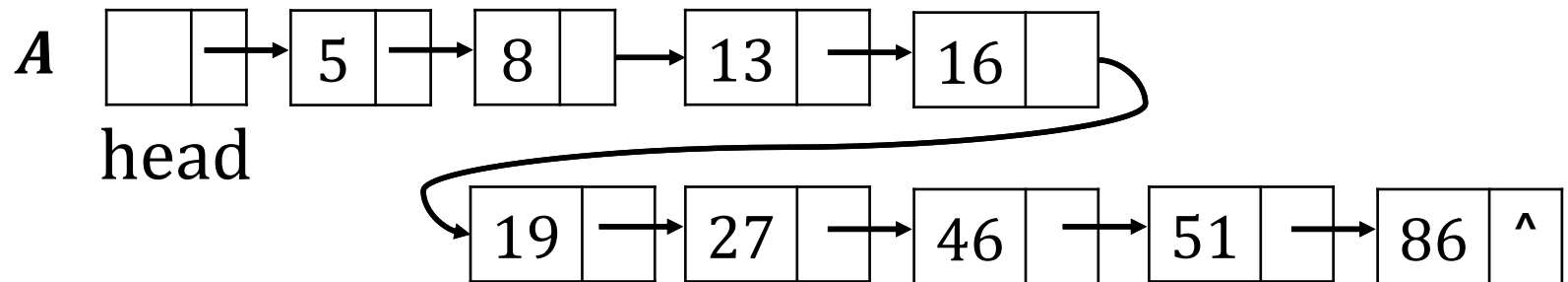


Features of Linked List

- ◆ It can grow without limit (not fixed length)
- ◆ Easy to insert/delete an element
- ◆ Delete 10 in Linked List A, before:

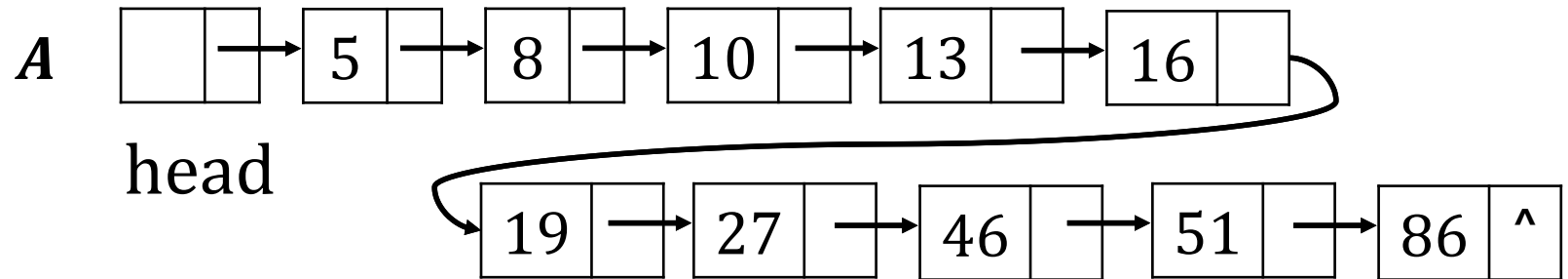


- ◆ After:

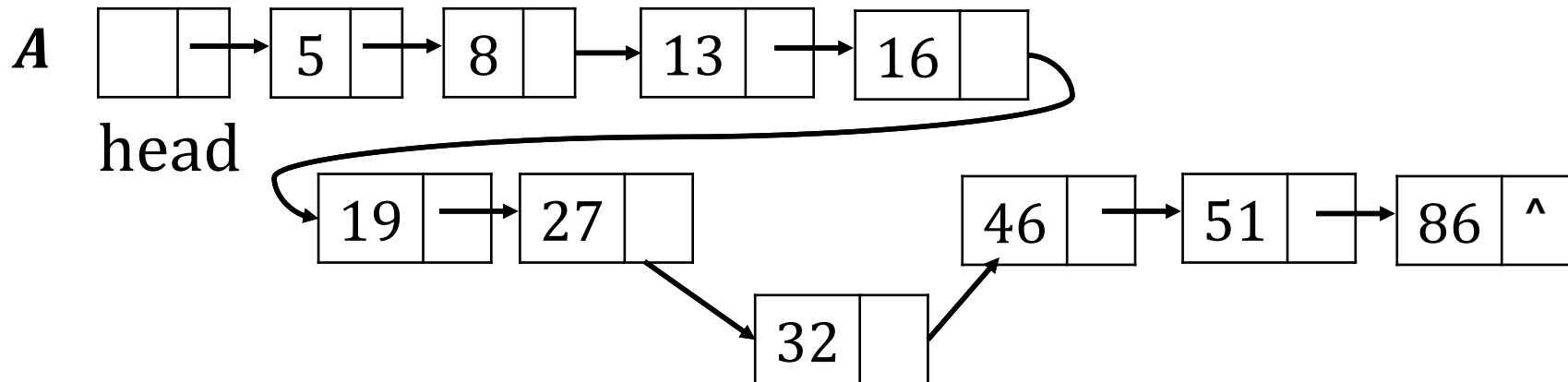


Features of Linked List

- ◆ Insert 32 in Linked List A, before:



- ◆ After:



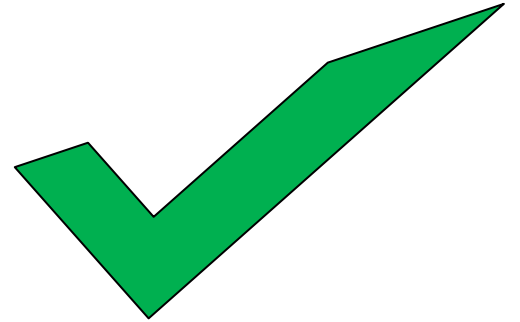
- ◆ Time Complexity?

Features of Linked List

- ◆ Disadvantages of Linked List
 - ◆ They do not provide random access
 - ◆ Need to “walk down” the list to access an item
 - ◆ The links take up additional memory
 - ◆ Not compact (in terms of Memory)
- ◆ Linked List vs. Array
 - ◆ Space complexity
 - ◆ Time Complexity: Insert, Delete, Find

Our Roadmap

- ◆ Linked List Definition
- ◆ Linked List Operators
- ◆ Illustration Example

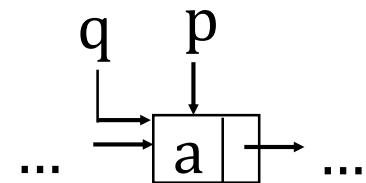
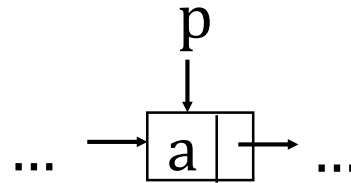


Basic Operators of Linked List

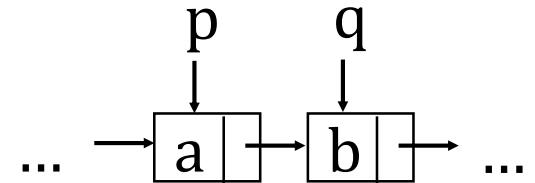
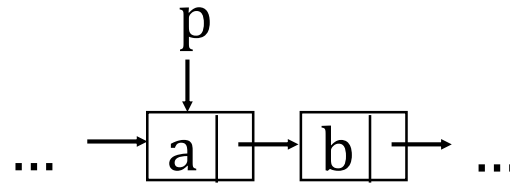
Before

After

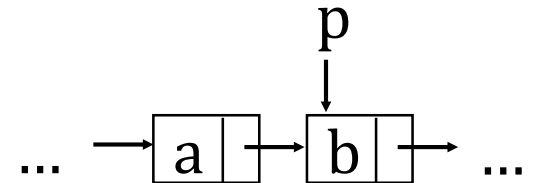
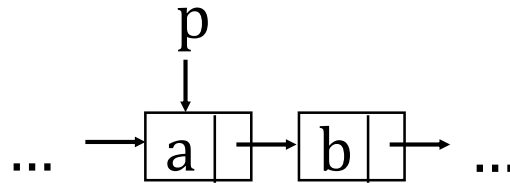
① $q \leftarrow p$



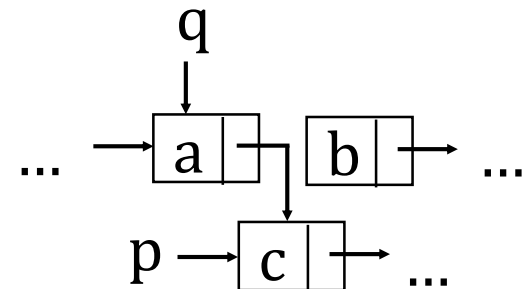
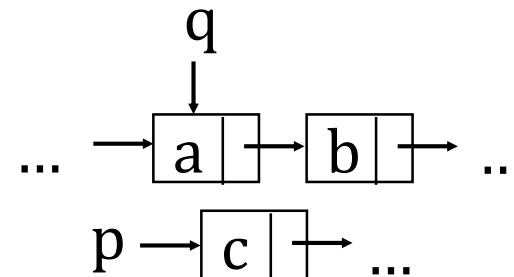
② $q \leftarrow \text{next of } p$



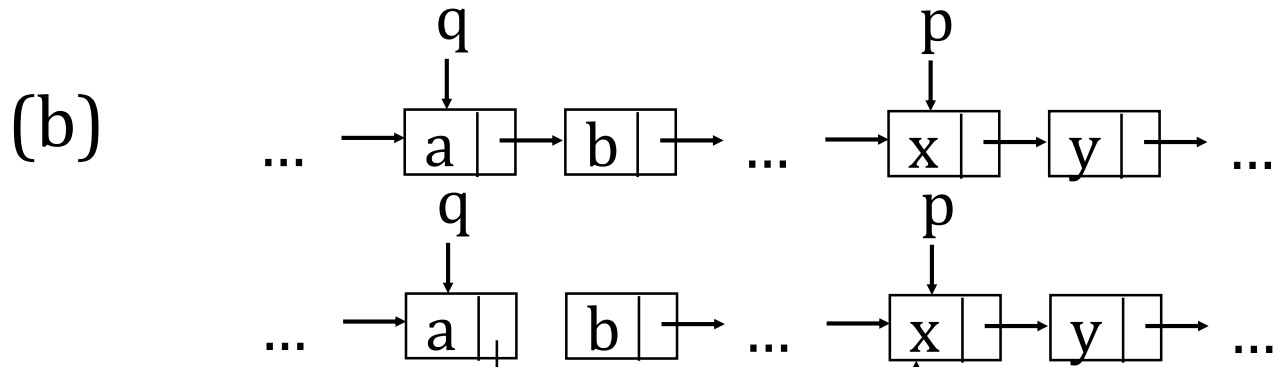
③ $p \leftarrow \text{next of } p$



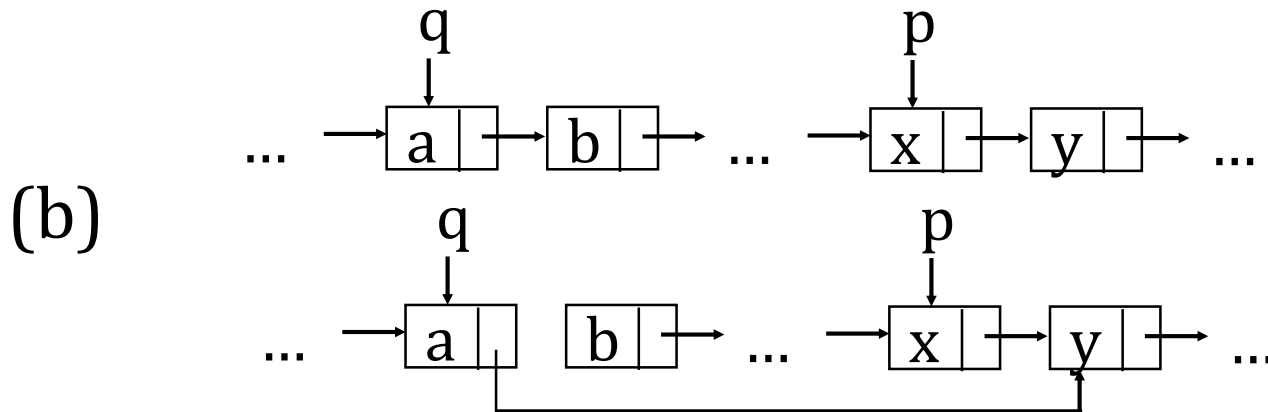
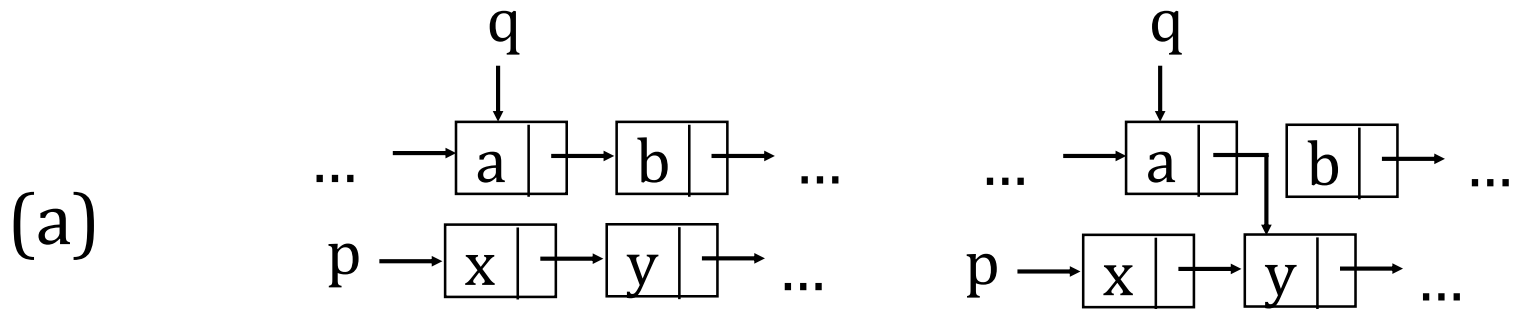
④ next of $q \leftarrow p$
(a)



Basic Operators of Linked List

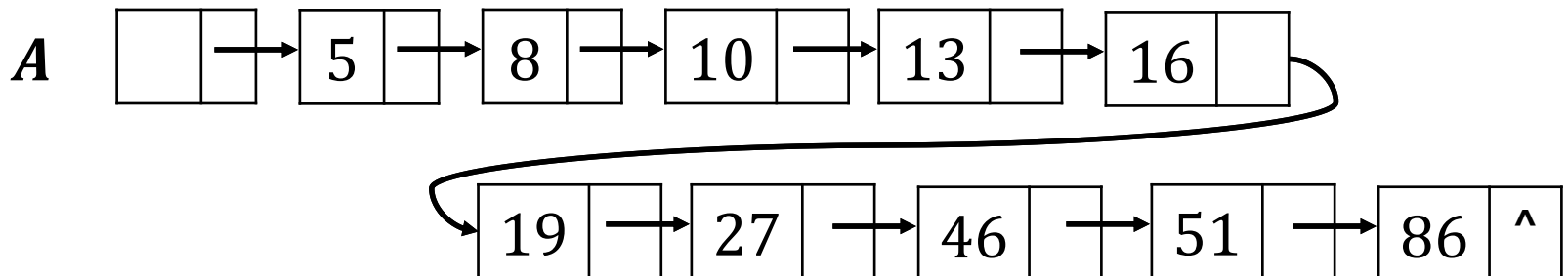


⑤ next of $q \leftarrow$ next of p



Traverse a Linked List

- ❖ Many tasks require us to traverse or “walk down” a linked list
- ❖ Recursion Pseudocode
- ❖ **Algorithm:** traverse(A):
 1. if (A=NULL)
 2. return
 3. else
 4. **print A.value**
 5. traverse(A.next)

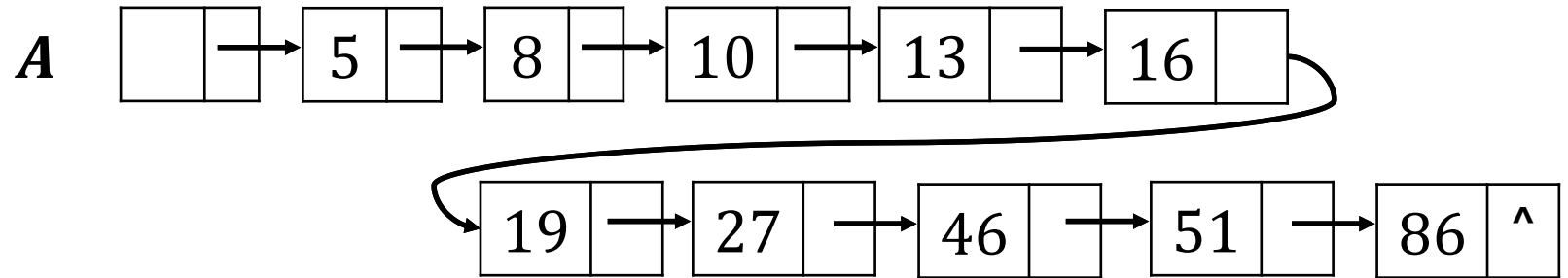


Traverse a Linked List

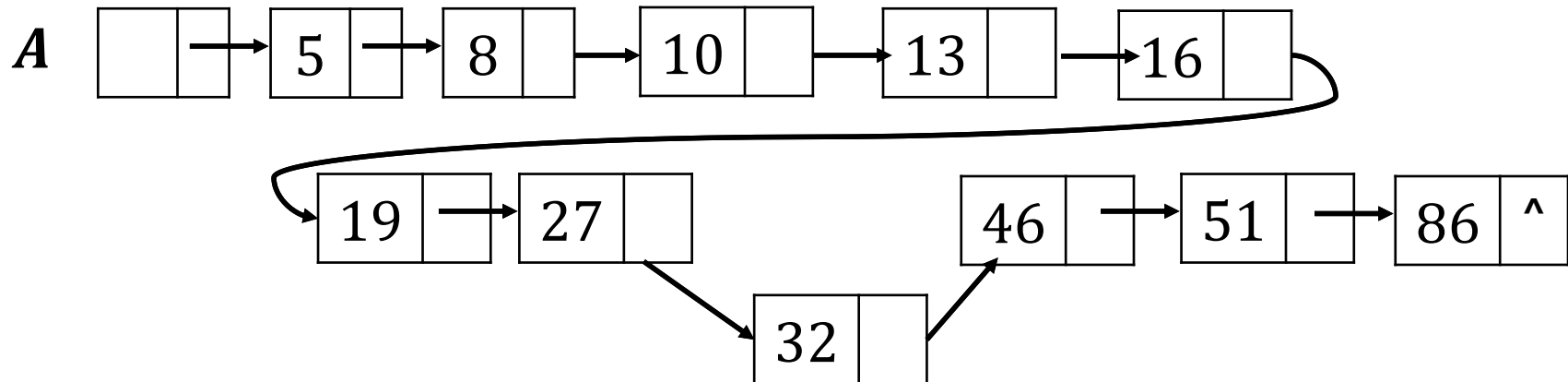
- ◆ It can also be done using iteration (for loops, while loops, etc.)
- ◆ Iteration Pseudocode
- ◆ **Algorithm:** traverseIteration(A):
 1. node trav \leftarrow A
 2. While (trav \neq NULL)
 3. **print** trav.value
 4. trav \leftarrow trav.next
- ◆ We use iteration in the following operators, but you can try to use recursion to implement these operators.

Inserting an Item at Position i

- ◆ Insert 32 in Linked List A at position 8, before:



- ◆ After:



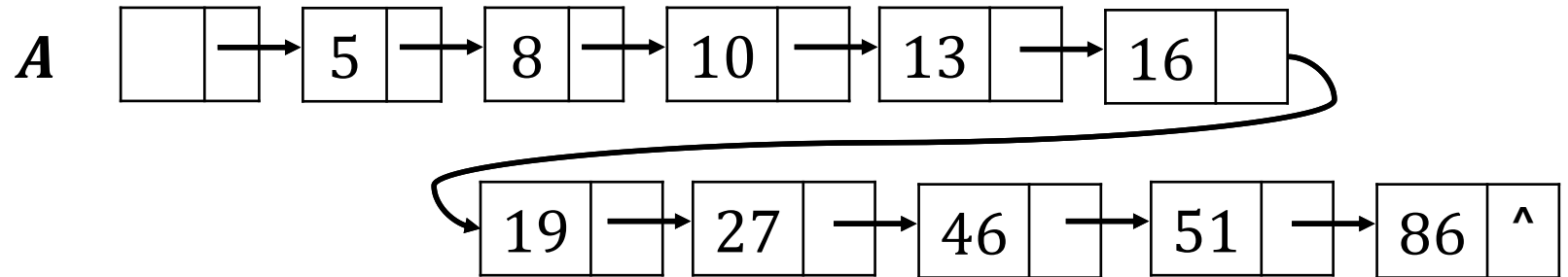
- ◆ How to do that?

Inserting an Item at Position i

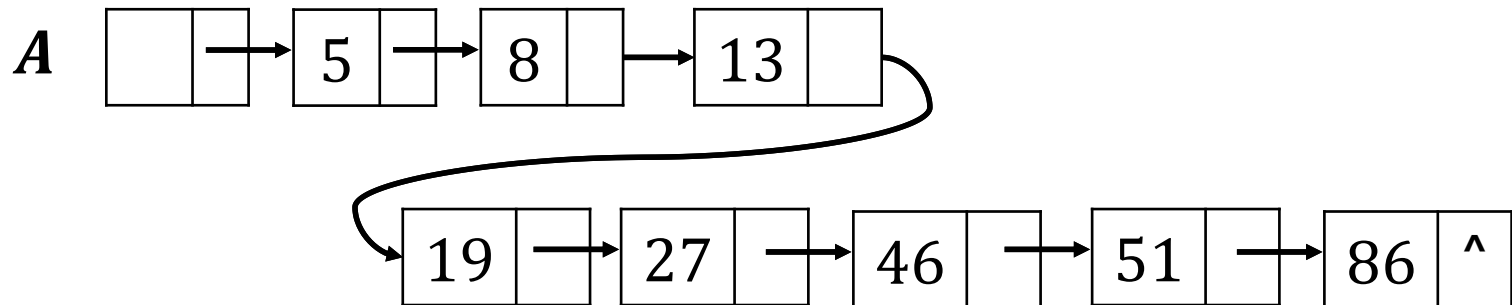
- ◆ **Problem:** insert node q in Linked List A at Position i
- ◆ **Algorithm:** insertNode(A, node q, i):
 1. $a \leftarrow 0$, node $p \leftarrow A$,
 2. **while** ($i-1 > a$)
 3. $p \leftarrow p.next$
 4. $a \leftarrow a + 1$
 5. $tmp \leftarrow p.next$
 6. $p.next \leftarrow q$
 7. $q.next \leftarrow tmp$
 8. **return** A
- ◆ Time Complexity: **$O(n)$**
- ◆ Space Complexity: **$O(1)$**

Deleting an Item at Position i

- ◆ Delete position 5 in Linked List A, before:



- ◆ After:



- ◆ How to do that?

Deleting an Item at Position i

- ◆ **Problem:** delete node in Linked List A at Position i
- ◆ **Algorithm:** deleteNode(A, i):
 1. $a \leftarrow 0$, node $p \leftarrow A$,
 2. **while** ($i-1 > a$)
 3. $p \leftarrow p.next$
 4. $a \leftarrow a + 1$
 5. $p.next \leftarrow p.next.next$
 6. **return** A
- ◆ Time Complexity: **$O(n)$**
- ◆ Space Complexity: **$O(1)$**

Finding an Item at Position i

◆ **Problem:** Find value x in Linked List A

◆ **Algorithm:** findValue(A, x):

```
1. a ← 0, node p ← A,  
2. while (p!=NULL)  
4.     if (x = p.value)  
5.         return p  
6.     p ← p.next  
7. return -1
```

◆ Time Complexity: **$O(n)$**

◆ Space Complexity: **$O(1)$**

Updating an Item at Position i

- ◆ **Problem:** Update nodes with value x to y in Linked List A
- ◆ **Algorithm:** updateNodes(A, x):
 1. $a \leftarrow \emptyset$, node $p \leftarrow A$,
 2. **while** ($p \neq \text{NULL}$)
 4. **if** ($x = p.\text{value}$)
 5. $p.\text{value} \leftarrow y$
 6. $p \leftarrow p.\text{next}$
 7. **return** A
- ◆ Time Complexity: **$O(n)$**
- ◆ Space Complexity: **$O(1)$**

Our Roadmap

- ◆ Linked List Definition
- ◆ Linked List Operators
- ◆ Illustration Example



Operators on polynomials

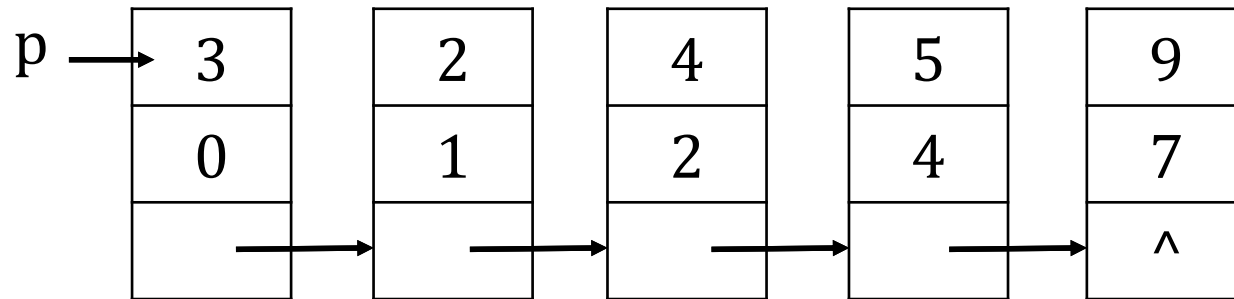
- ◆ **Polynomials:** $p(x) = p_0 + p_1x + p_2x^2 + \dots + p_nx^n$
- ◆ a set of ordered pairs of $\langle p_i, i \rangle$ where p_i is the coefficient and i is the exponent.
- ◆ We use linked list store the $\langle p_i, i \rangle$ pairs of $p(x)$
- ◆ Without loss of generality, we skip all nodes w/ $p_i = 0$
- ◆ Node representation:

```
node polyItem{  
    float coef    // record  $p_i$   
    int  expo    // record exponent  
    node next    // reference to next polyItem  
}
```

- ◆ **Question:** how about use array?

Finding degree of a Polynomials

◆ **Polynomials:** $p(x) = 3 + 2x + 4x^2 + 5x^4 + 9x^7$



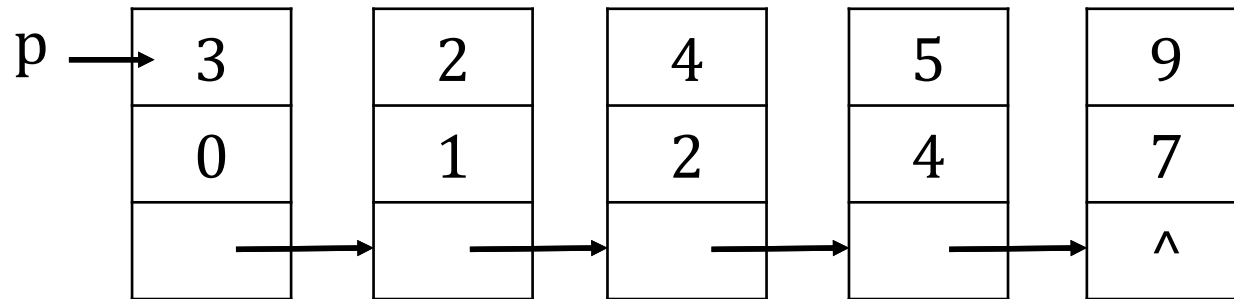
◆ Degree of $p(x)$: 7

◆ **Algorithm:** findDegree(p):

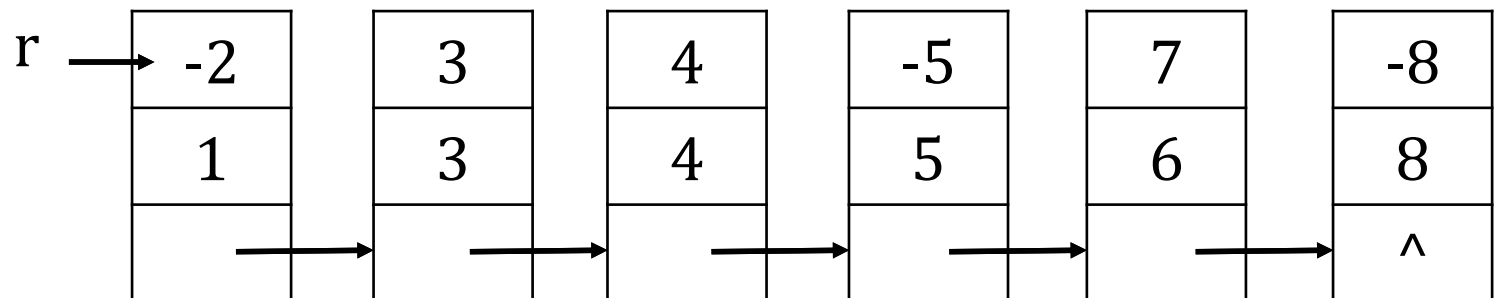
1. node tmp \leftarrow p
2. **While** (tmp.next \neq NULL)
3. tmp \leftarrow tmp.next
4. **return** tmp.expo

Adding two polynomials

◆ $p(x) = 3 + 2x + 4x^2 + 5x^4 + 9x^7$

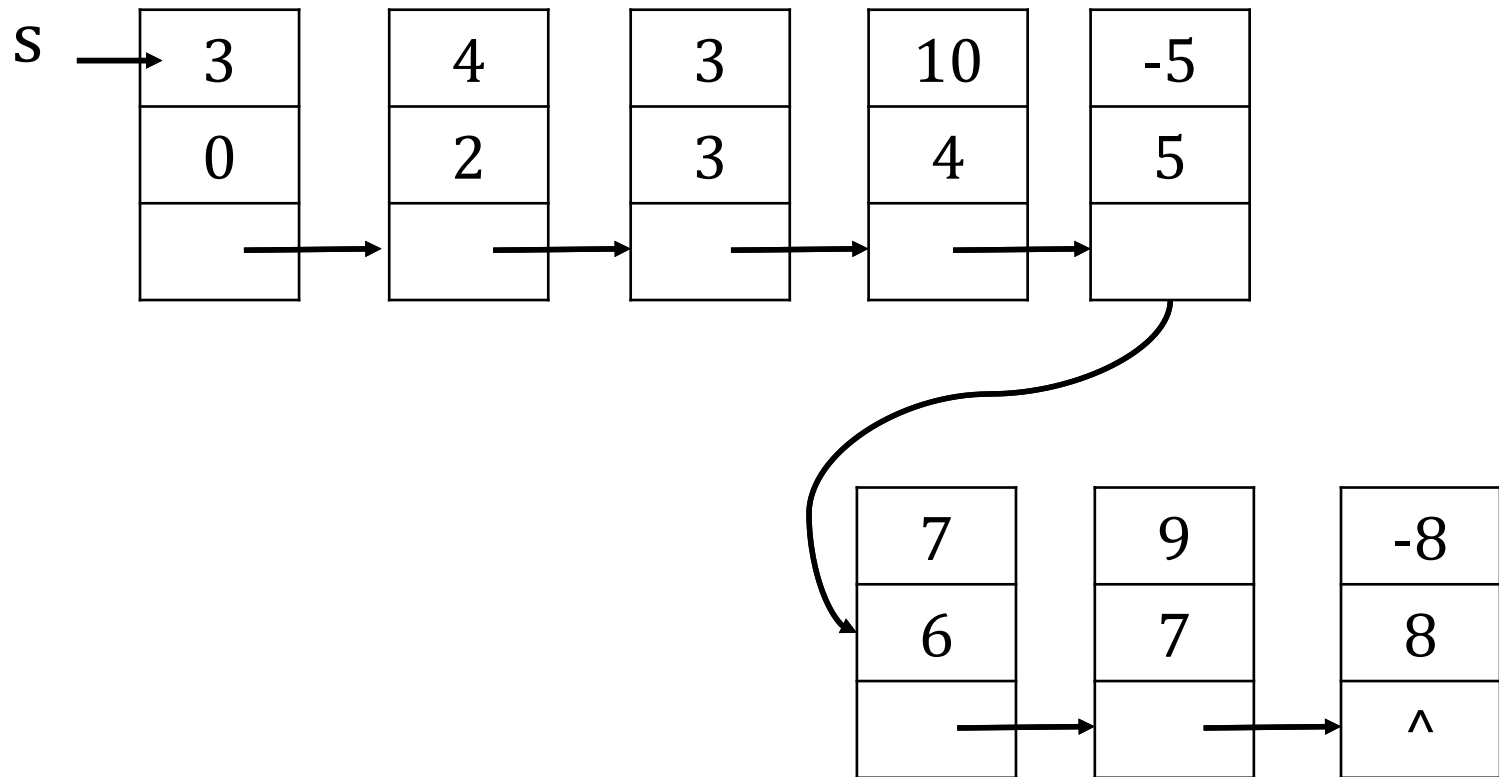


◆ $r(x) = -2x + 3x^3 + 5x^4 - 5x^5 + 7x^6 - 8x^8$



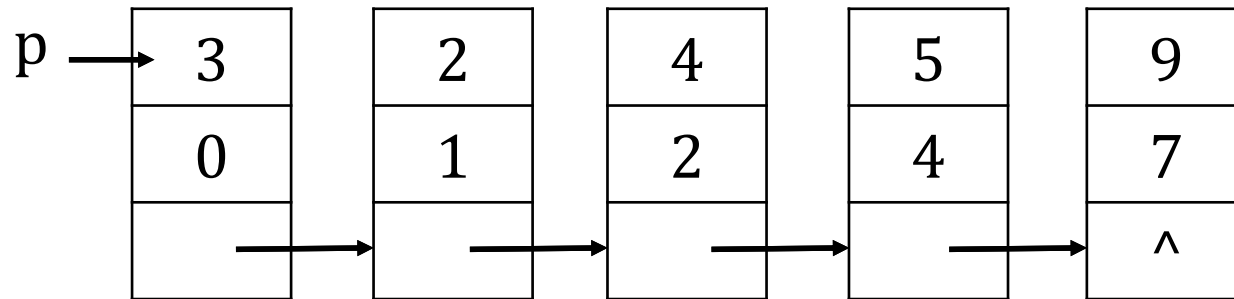
Adding two polynomials

◆ $s(x) = p(x) + r(x)$
 $= 3 + 4x^2 + 3x^3 + 10x^4 - 5x^5 + 7x^6 + 9x^7 - 8x^8$

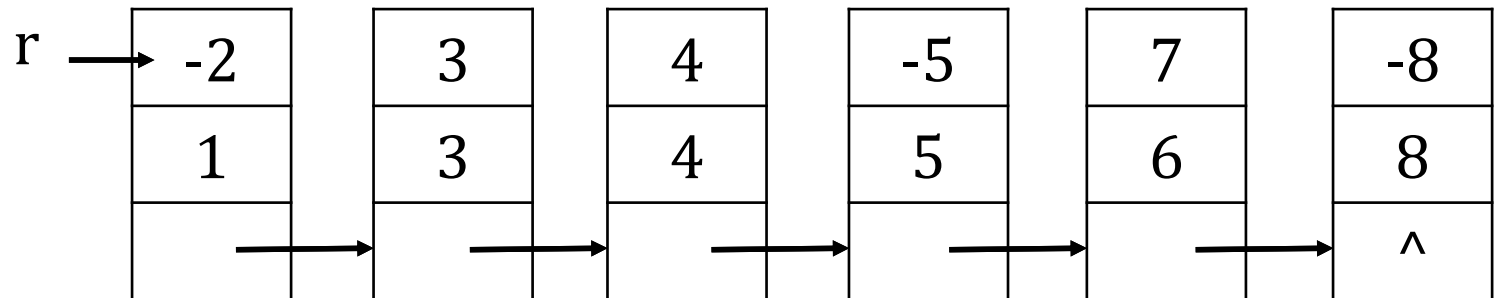


Subtracting two polynomials

◆ $p(x) = 3 + 2x + 4x^2 + 5x^4 + 9x^7$

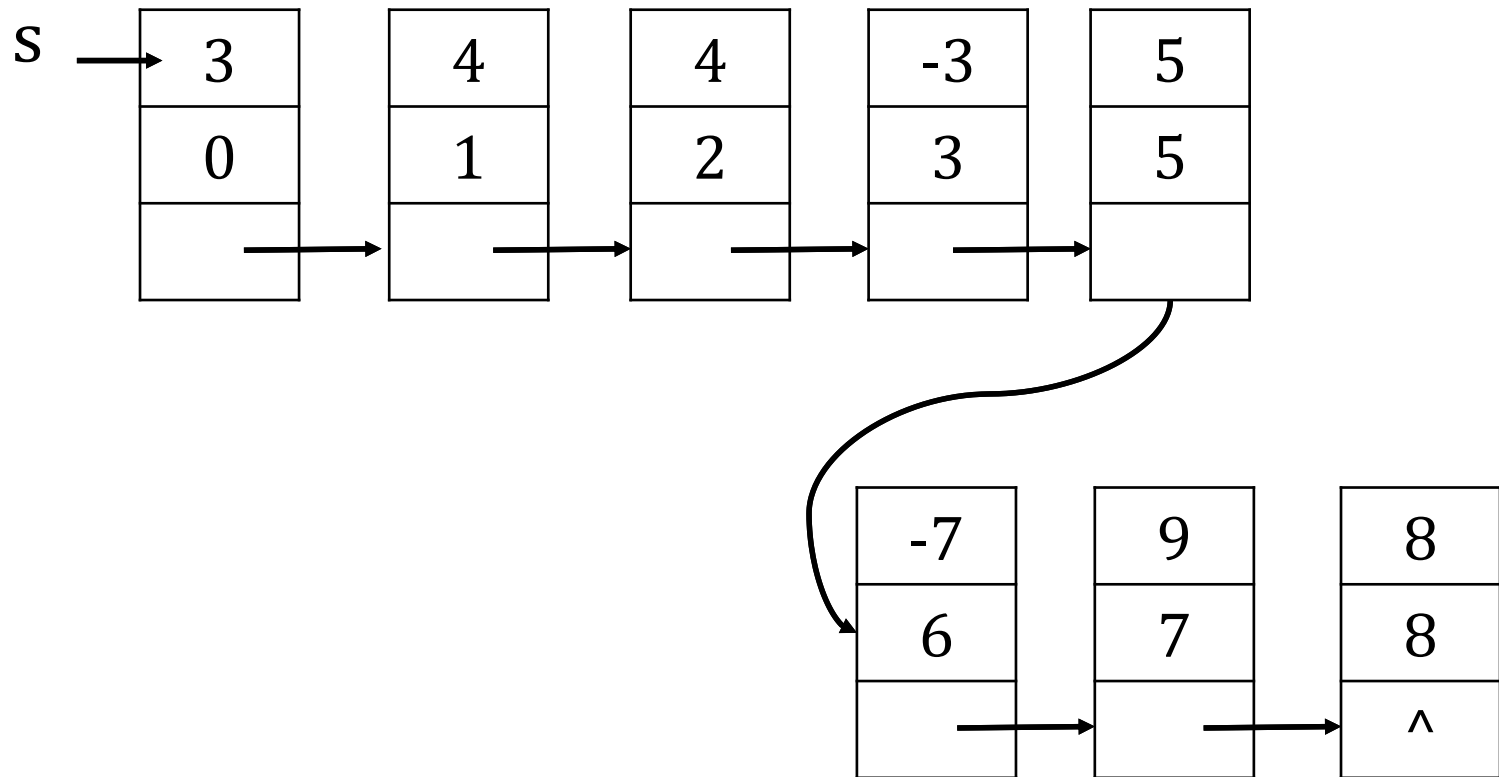


◆ $r(x) = -2x + 3x^3 + 5x^4 - 5x^5 + 7x^6 - 8x^8$



Subtracting two polynomials

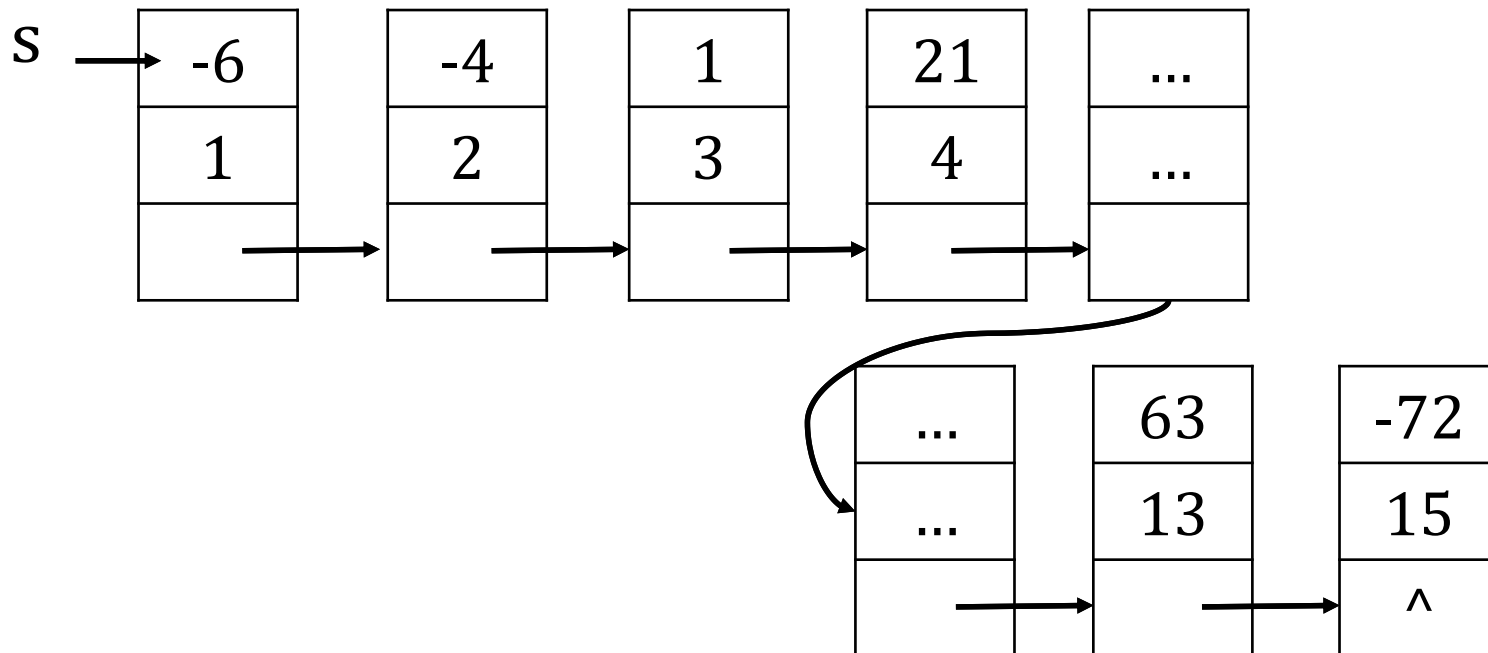
$\diamond s(x) = p(x) - r(x)$
 $= 3 + 4x + 4x^2 - 3x^3 + 5x^5 - 7x^6 + 9x^7 + 8x^8$



Multiplying two polynomials

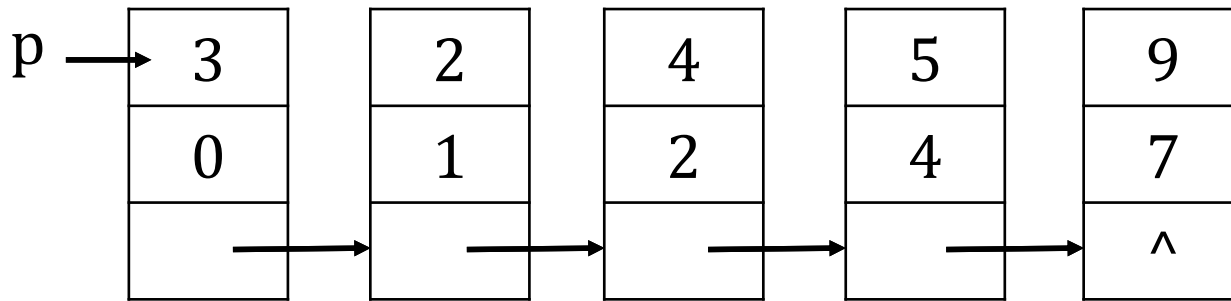
- ◇ $p(x) = 3 + 2x + 4x^2 + 5x^4 + 9x^7$
- ◇ $r(x) = -2x + 3x^3 + 5x^4 - 5x^5 + 7x^6 - 8x^8$
- ◇ $s(x) = p(x) * r(x)$

$$= -6x - 4x^2 + x^3 + 21x^4 - 3x^5 + 31x^6 + 9x^7 + 11x^8 - 41x^9 + 30x^{10} + 45x^{11} - 85x^{12} + 63x^{13} - 72x^{15}$$

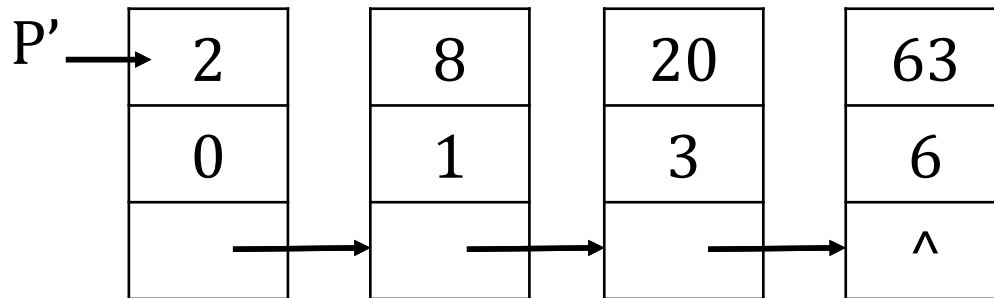


Differentiating of a polynomial

◆ $p(x) = 3 + 2x + 4x^2 + 5x^4 + 9x^7$

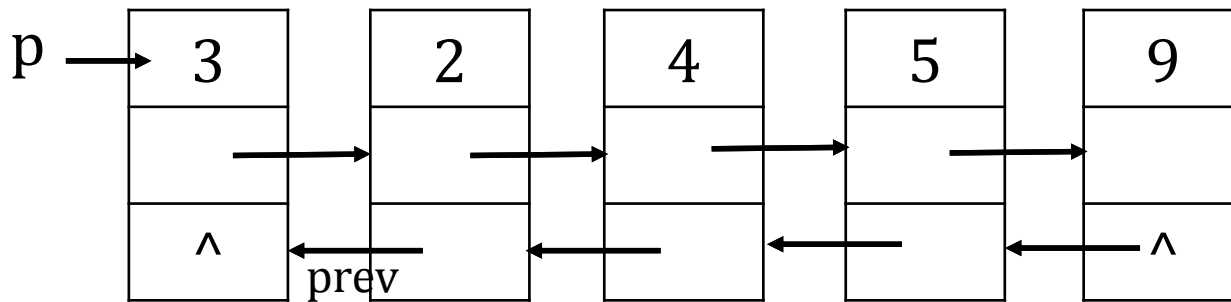


◆ $p'(x) = 2 + 8x + 20x^3 + 63x^6$



Other variants of Lined List

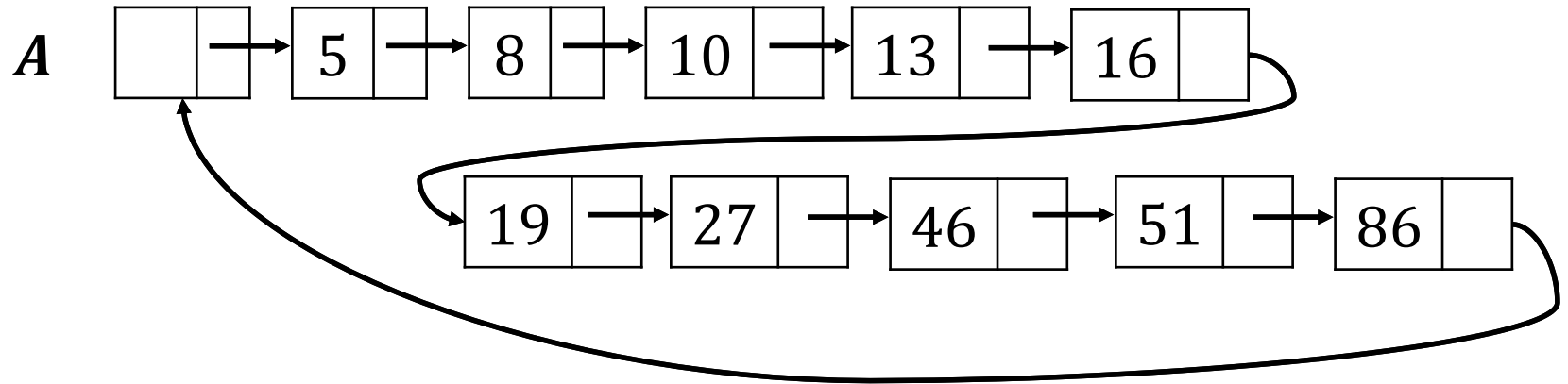
◆ Double linked list



- ◆ add a prev reference to each node: refers to the previous node
- ◆ allow us to “back up” from a given node

Other variants of Lined List

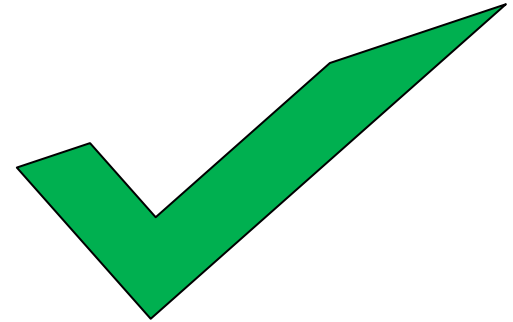
◆ Circular linked list



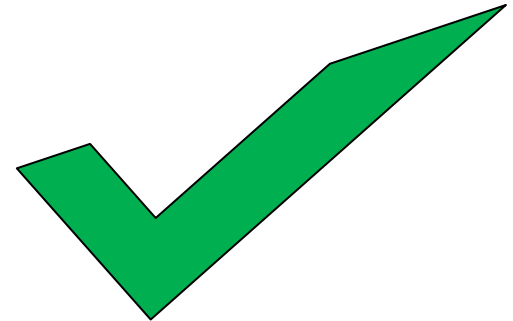
- ◆ Is it a empty list? $\text{head.next} = \text{head}$?
- ◆ Is it the end of list? $\text{tmp.next} = \text{head}$?

Our Roadmap

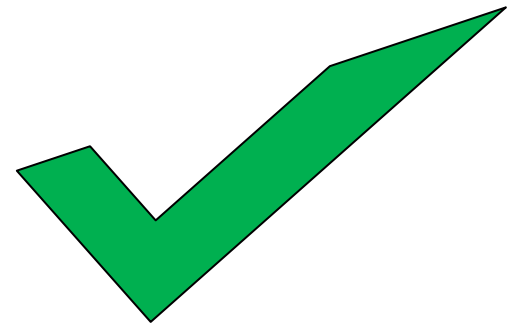
- ◆ Linked List Definition



- ◆ Linked List Operators



- ◆ Illustration Example



Thank You!